

RĪGAS VALSTS TEHNIKUMS

DATORIKAS NODAĻA

Izglītības programma: Programmēšana

KVALIFIKĀCIJAS DARBS

“Elektropreču internetveikala datu uzskaites sistēma”

Paskaidrojošais raksts 111 lpp.

Audzēknis:

Kristaps Ešmits

Prakses vadītājs:

Ilona Demčenko

Nodaļas vadītājs:

Normunds Barbāns

Rīga 2024

ANOTĀCIJA

Kvalifikācijas darba uzdevums ir izveidot “Elektropreču internetveikala datu uzskaites sistēmu”, ar kuru jebkurš cilvēks spēs pilnvērtīgi darboties neatkarīgi vai ir apmācīts, vai nav apmācīts darboties ar konkrētos sistēmu. Sistēma tiek veidota pēc konkrēta uzņēmuma pasūtījuma. Backend daļas izstrādei tika izvēlēts PHP, Laravel, Axios. Servera daļai tiek izmantots XAMPP, Apache. Lietotāja saskarnei tika izvēlēts Vue.js, Vite.js, Bootstrap. Priekš datu bāzes tiek izmantots MySQL ar phpmyadmin.

Kvalifikācijas darba paskaidrojošais raksts satur – ievadu, uzdevuma nostādni, prasību specifikāciju, uzdevuma risināšanas līdzekļu izvēles pamatojumu, programmatūras produkta modelēšanas un projektēšanas aprakstu, datu struktūru aprakstu, lietotāja ceļvedi, nobeigumu un pielikumus. Ievadā ir aprakstīts sistēmas nepieciešamība, aktualitāte un lietderīgums. Uzdevuma nostādnē ir aprakstīts kvalifikācijas darba galvenais izveidošanas mērķis un tā uzdevumi. Prasību specifikācijā ir aprakstītas sistēmas funkcionālās un nefunkcionālās prasības, kā arī aplūkojama sistēmas izejas un ieejas informācija – dati, ko lietotājs ievada un dati, kuri tiek izvadīti lietotājam. Uzdevuma risināšanas līdzekļu izvēles pamatojumā ir aprakstīts, kādas programmēšanas valodas, teksta redaktori, datu bāzes, rīki tika izmantoti sistēmas izstrādē. Programmatūras produkta modelēšanas un projektēšanas aprakstā ir apskatāma datu plūsmu diagramma, ER diagramma, datu bāzes uzbūve, kā arī aprakstīta sistēmas arhitektūra. Datu struktūru apraksts satur visu tabulu struktūru un to aprakstu, kā arī tabulu relāciju shēmu. Lietotāja ceļvedis detalizēti attēlo sistēmas informācijas vizuālo izkārtojumu un paskaidro kā pareizi lietot sistēmu. Nobeigumā ir aprakstīts, kas beigās tika izveidots.

Kvalifikācijas darba apjoms ir 111 lpp, kurā ietilps 70 attēli, 9 tabulas un 3 pielikumi.

ANNOTATION

The task of the qualification work is to create a "data accounting system of an electrical goods online store", with which any person will be able to fully function, regardless of whether he is trained or not trained to operate the specific system. The system is created according to the order of a specific company. PHP, Laravel, Axios were chosen for the development of the backend part. XAMPP, Apache is used for the server part. Vue.js, Vite.js, Bootstrap were chosen for the user interface. MySQL with php myadmin is used for the database.

The explanatory article of the qualification work contains – introduction, statement of the task, specification of requirements, justification for choosing means of solving the task, description of software product modelling and design, description of data structures, users guide, conclusion and appendices. The introduction describes the need, relevance and usefulness of the system. The assignment statement describes the main purpose of creating the qualification work and its tasks. The requirements specification describes the functional and non-functional requirements of the system, as well as the output and input information of the system - data that is entered by the user and data that is output to the user. The justification for choosing the tools for solving the task describes what programming languages, text editors, databases, and tools were used in the development of the system. The software product modelling and design description includes a data flow diagram, an ER diagram, a database structure, as well as a described system architecture. The description of data structures contains the structure of all tables and their description, as well as the relational schema of the tables. The user guide shows the visual layout of the system information in detail and explains how to use the system correctly. The conclusion describes what was created in the end.

The scope of the qualification work is 111 pages, which includes 70 images, 9 tables and 3 appendices.

SATURS

IEVADS.....	5
1. UZDEVUMA NOSTĀDNE	6
2. PRASĪBU SPECIFIKĀCIJA	7
2.1. Ieejas un izejas informācijas apraksts.....	7
2.1.1. Ieejas informācijas apraksts.....	7
2.1.2. Izejas informācijas apraksts.....	10
2.2. Funkcionālās prasības.....	10
2.3. Nefunkcionālās prasības.....	13
3. UZDEVUMA RISINĀŠANAS LĪDZEKĻU IZVĒLES PAMATOJUMS	15
4. PROGRAMMATŪRAS PRODUKTA MODELĒŠANA UN PROJEKTĒŠANA	16
4.1. Sistēmas struktūras modelis	16
4.1.1. Sistēmas arhitektūra.....	16
4.1.2. Sistēmas ER-modelis	18
4.2. Funkcionālais sistēmas modelis.....	19
4.2.1. Datu plūsmu modelis.....	19
5. DATU STRUKTŪRU APRAKSTS	24
6. LIETOTĀJA CEĻVEDIS	30
6.1. Sistēmas prasības aparatūrai un programmatūrai	30
6.2. Sistēmas instalācija un palaišana.....	30
6.3. Programmas apraksts.....	34
6.4. Testa piemērs.....	54
NOBEIGUMS	56
INFORMĀCIJAS AVOTI.....	57
PIELIKUMI.....	58
1. pielikums. Pirkuma PDF izdruka.....	59
2. pielikums. Preces specifikācijas XLS izdruka.....	60
3. pielikums. Programmas pirmteksts.....	61

IEVADS

Mūsdienās ir ļoti svarīgi izmantot modernas datu uzskaites sistēmas, jo informācijas un digitālās tehnoloģijas attīstās straujāk nekā jebkad, tāpēc ir nepieciešamas sistēmas, kas atbilst mūsdienu standartiem, ir viegli pielāgojamas un ātrdarbīgas. Šī kvalifikācijas darba mērķis ir izveidot “Elektropreču internetveikala datu uzskaites sistēmu”, ar kuru jebkurš cilvēks spēs pilnvērtīgi darboties neatkarīgi vai ir apmācīts, vai nav apmācīts darboties ar konkrētos sistēmu. Sistēma tiek veidota pēc konkrēta uzņēmuma pasūtījuma, bet to ir iespējams arī pielāgot citiem uzņēmumiem.

Pirms datu uzskaites sistēmas izstrādes tiek veikta tirgus izpēte, lai saprastu, kādi produkti jau atrodas pašreizējā tirgū, to trūkumi un priekšrocības, kā arī kāpēc pasūtītājs vēlas pilnīgi jaunu datu uzskaites sistēmu, nevis kādu no tirgū jau esošajām datu uzskaites sistēmām. Esošie analogi klientu neapmierini, jo tie ir ļoti dārgi, it īpaši biznesam paplašinoties. Lai sasniegtu pasūtītāja velmes būtu nepieciešams implementēt divas dažādas jau esošas sistēmas. Viena, kas atbildētu par noliktavas pārvaldību, otra kas atbildētu par internetveikala darbību, tādējādi ļoti sadārdzinot izmaksas. Problēma izmantojot pašreizējos analogus ir tāda, ka ir nepieciešams apmācības periods, kā arī tie neatbilst līdz galam pasūtītāja velmēm. Viens no analoģu piemēriem ir “SAP Inventory Management”. Šī sistēma atbilst daudziem no pasūtītāja punktiem - automatizēta noliktavas darbība, analītiskie rīki, bet tai ir augstas izmaksas, sarežģīta implementācija, sarežģītā pielāgošana un ilgs apmācību periods, kā arī tā atbild tikai par noliktavas daļu, tas nozīmē, ka ir nepieciešams implementēt vēl vienu sistēmu, kā ” Magento”, kura atbildēs par internetveikala darbību.

Pašreizējai datu uzskaites sistēmai pasūtītājs izmanto Excel, tādējādi bieži vien uzņēmums saskaras ar ierobežojumiem, kas ietekmē uzņēmumu efektivitāti un attīstību. Uzņēmumam augot datu apjoms palielinās, tādējādi ir grūti apkopot visus datus, kā arī Excel pie liela datu apjoma kļūst lēns, tādējādi datu apstrādes laiks saīdzinās un pat rodas neprecizitātes.

Izmantojot “Elektropreču internetveikala datu uzskaites sistēmu” pasūtītājs atvieglo dzīvi darbiniekiem, jo sistēmas izmantošanai nevajag apmācību, tā nodrošina visu nepieciešamos līdzekļus, kā arī ietaupa līdzekļus, ja salīdzina ar tirgū esošajiem analogiem.

1. UZDEVUMA NOSTĀDNE

Kvalifikācijas darba uzdevums ir izveidot Elektropreču internetveikala datu uzskaites sistēmu. Sistēmā nepieciešams realizēt iespēju sistēmas administratoram veikt uzskaiti un darbības ar precēm, kuras pieder veikalam, kā arī radīt iespēju potenciālajiem pircējiem iespēju aplūkot un iegādāties kādu no precēm.

Elektropreču internetveikala datu uzskaites sistēma ir aktuāla konkrētam pasūtītājam, lai uzlabotu sava veikala, noliktavas darbību, kā arī veikala apmeklētājiem, tādējādi dodot tiem iespēju iepirkties un uzzināt informāciju gan par precēm, gan par veikalu neizejot no mājas. Pašreizējie tirgū pieejamie interneta veikali neefektīvi prezentē administratīvos datus un klientu informāciju. Sistēma radītu intuitīvāku sadarbību starp klientiem un administratoriem caur skaidrāku datu vizualizāciju. Mērķauditorija ir ne tikai konkrētais pasūtītājs un tā klientu loks, bet arī citi uzņēmumi, kuriem ir iespējams piemērot šādu uzskaites sistēmu.

Elektropreču internetveikala datu uzskaites sistēmai ir jāizpilda sekojošas funkcionalitātes (skat. 1.1. att.):

- preču pievienošana, labošana, dzēšana;
- pasūtījumu veikšana;
- preču meklēšana;
- preču filtrēšana pēc parametriem;
- preču pieejamības pārbaude;
- lietotāja reģistrēšana, autorizēšana, datu labošana, profila apskate.



1.1. att. Lietojumgadījuma diagramma

2. PRASĪBU SPECIFIKĀCIJA

2.1. Ieejas un izejas informācijas apraksts

2.1.1. Ieejas informācijas apraksts

Sistēmā tiks nodrošināta šādas ieejas informācijas apstrāde:

1. Informācija par **Lietotājiem** sastāvēs no šādiem datiem.

- Vārds – lietotāja vārds – burtu teksts ar izmēru līdz 255 rakstzīmēm, piemēram - Jānis;
- Uzvārds – lietotāja uzvārds – burtu teksts ar izmēru līdz 255 rakstzīmēm, piemēram - Bērziņš;
- Parole – parole, ko lietotājs izmantos, lai ienāktu sistēmā – burtu teksts ar izmēru līdz 255 rakstzīmēm, piemēram - JanisB1987;
- Telefona numurs – lietotāja telefona numurs – cipars ar izmēru līdz 15 rakstzīmēm, piemēram 25677359;
- E-pasts – lietotāja e-pasts – burtu teksts ar izmēru līdz 255 rakstzīmēm, piemēram - janisberzins@inbox.lv;
- Loma – lietotāja loma, viesis, administrators, vai reģistrēts lietotājs automātiski reģistrēts lietotājs – cipars ar izmēru līdz 1 rakstzīmei, piemēram - 1;
- Izveidots – automātiska aile ar datumu un laiku, piemēram - 2023-11-02 14:29:13;
- Atjaunots – automātiska aile ar datumu un laiku, piemēram - 2023-11-02 14:29:13.

* Visu informāciju par **Lietotājiem** ievadīs lietotājs no klaviatūras izņemot laukus izveidots, atjaunināts un loma, tos iegūs no API.

2. Informācija par **Kategorijām** sastāvēs no šādiem datiem.

- Nosaukums – Kategorijas nosaukums – burtu teksts ar izmēru līdz 50 rakstzīmēm, piemēram - Klaviatūras;
- Izveidots – automātiska aile ar datumu un laiku, piemēram - 2023-11-02 14:29:13;
- Atjaunots – automātiska aile ar datumu un laiku, piemēram - 2023-11-02 14:29:13.

* Visu informāciju par **Kategorijām** ievadīs lietotājs no klaviatūras, izņemot laukus izveidots un atjaunināts, tos iegūs no API.

3. Informācija par **Precēm** sastāvēs no šādiem datiem.

- Nosaukums – preces nosaukums – burtu teksts ar izmēru līdz 255 rakstzīmēm, piemēram- Klaviatūra mikro pele;
- Apraksts – preces apraksts – burtu teksts ar izmēru līdz 255 rakstzīmēm, piemēram - Jaunākā tipa klaviatūra ar visām nepieciešamajām funkcijām;
- Cena – preces cena – daļskaitlis ar precizitāti līdz 2 cipariem aiz komata, piemēram - 90.99;
- Nodoklis – preces pievienotās vērtības nodoklis – cipars līdz 11 rakstzīmēm, pēc noklusējuma 21, piemēram - 5;
- Bilde – preces bilde – burtu teksts ar izmēru līdz 255 rakstzīmēm, piemēram - EVk7LOgI2G.png;
- Daudzums – preces daudzums, kāds pieejams – cipars līdz 11 rakstzīmēm, piemēram – 100;
- Rezervēts – preces daudzums, kāds ir rezervēts – cipars līdz 11 rakstzīmēm, piemēram – 50;
- Pārdots – preces daudzums, kāds ir pārdots – cipars līdz 11 rakstzīmēm, piemēram – 20;
- Izveidots – automātiska aile ar datumu un laiku, piemēram - 2023-11-02 14:29:13;
- Atjaunots – automātiska aile ar datumu un laiku, piemēram - 2023-11-02 14:29:13.

* Visu informāciju par **Precēm** ievadīs lietotājs no klaviatūras, izņemot laukus izveidots, atjaunināts, tos iegūs no API.

4. Informācija par **Zīmoliem** sastāvēs no šādiem datiem.

- Nosaukums – zīmola nosaukums – burtu teksts ar izmēru līdz 255 rakstzīmēm, piemēram – Klaviatūra mikro pele;
- Bilde – zīmola logo – burtu teksts ar izmēru līdz 255 rakstzīmēm, piemēram - EVk7LOgI2G.png;
- Izveidots – automātiska aile ar datumu un laiku, piemēram - 2023-11-02 14:29:13;
- Atjaunots – automātiska aile ar datumu un laiku, piemēram - 2023-11-02 14:29:13.

* Visu informāciju par **Zīmoliem** ievadīs lietotājs no klaviatūras, izņemot laukus izveidots, atjaunināts, tos iegūs no API.

5. Informācija par **Pirkumi** sastāvēs no šādiem datiem.

- Kopējā cena – kopējā cena, cik par precēm ir jāmaksā – daļskaitlis ar precizitāti līdz 2 cipariem aiz komata, piemēram - 90.99;
- Status – Pasūtījuma status vai pasūtījums ir izpildīts vai nē – burtu teksts ar izmēru līdz 255 rakstzīmēm, piemēram closed;
- Daudzums – iegādātās preces daudzums – cipars līdz 11 rakstzīmēm, piemēram 20;
- Nodoklis – preces pievienotās vērtības nodoklis – cipars līdz 11 rakstzīmēm, pēc noklusējuma 21, piemēram - 5;
- Izveidots – automātiska aile ar datumu un laiku, piemēram - 2023-11-02 14:29:13;
- Atjaunots – automātiska aile ar datumu un laiku, piemēram - 2023-11-02 14:29:13.

* Visu informāciju par **Grozu** iegūst no API.

6. Informācija par **Specifikāciju tiptiem** sastāvēs no šādiem datiem.

- Nosaukums – specifikācijas tipa nosaukums – burtu teksts ar izmēru līdz 255 rakstzīmēm, piemēram – Krāsa;
- Izveidots – automātiska aile ar datumu un laiku, piemēram - 2023-11-02 14:29:13;
- Atjaunots – automātiska aile ar datumu un laiku, piemēram - 2023-11-02 14:29:13.

* Visu informāciju par **Specifikāciju tiptiem** ievadīs lietotājs no klaviatūras, izņemot laukus izveidots, atjaunināts, tos iegūs no API.

6. Informācija par **Specifikāciju aprakstiem** sastāvēs no šādiem datiem.

- Apraksts – specifikācijas apraksts – burtu teksts ar izmēru līdz 255 rakstzīmēm, piemēram – Zila;
- Izveidots – automātiska aile ar datumu un laiku, piemēram - 2023-11-02 14:29:13;
- Atjaunots – automātiska aile ar datumu un laiku, piemēram - 2023-11-02 14:29:13.

* Visu informāciju **Specifikāciju aprakstiem** ievadīs lietotājs no klaviatūras, izņemot laukus izveidots, atjaunināts, tos iegūs no API.

2.1.2. Izejas informācijas apraksts

1. **Kvīts izvade PDF formātā** pēc pasūtījuma veikšanas. PDF failā tiks atspoguļota informācija par veikto iterāciju. Augšā pa vidu būs firmas logo un nosaukums, kreisajā pusē būs norādīts pirkuma datums, faila vidū būs tabula ar detalizētu pirkuma informāciju (preces nosaukums, cena, cik samaksāts). Dokumenta apakšā kreisajā pusē būs vieta kur parakstīties izsniedzējam un labajā pusē būs vietas kur parakstīties saņēmējam.
2. **Statistikas paziņojumi.** Statistikas paziņojumi būs pieejams administratoram, lai uzzinātu informāciju par sistēmu (Cik ir reģistrēti lietotāji, cik preces sistēmā u.t.t.).
3. **Datu izvade no administrēšanas sistēmas Excel formātā.** Administratoram būs iespēja izvadīt datus no administrēšanas rīka Excel formātā, ja būs tāda nepieciešamība. Būs iespējams izvadīt visu tabulu vai nepieciešamos laukus.

2.2. Funkcionālās prasības

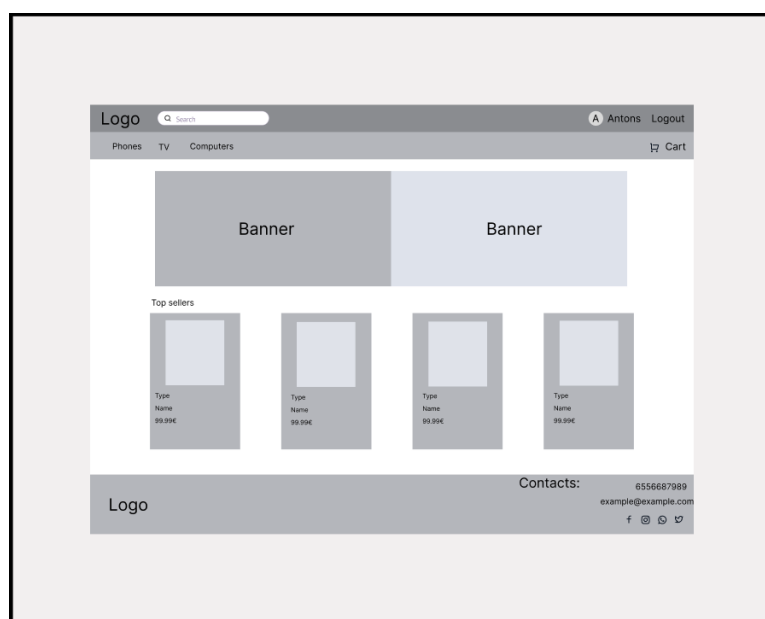
1. Jānodrošina iespēja reģistrēt jaunu lietotāju.
 - 1.1. Jāparedz ieejas informācijas par lietotāju (skat. 2.1.1. Ieejas informācijas apraksts) ievadīšana un pārbaude uz formāta pareizību.
 - 1.2. Ja kāds no obligātiem laukiem nav ievadīts, tad izvadīt par to kļūdas paziņojumu.
 - 1.3. Salīdzināt ievadīto e-pastu, telefona numuru ar sistēmā jau eksistējošo lietotāju vārdiem un izvadīt paziņojumu, ja tie sakrīt.
2. Jānodrošina lietotāja autorizācija.
 - 2.1. Ja lietotāja statuss ir aktīvs sistēmai ir jānodrošina autorizācija, pieslēdzoties ar e-pastu un paroli.
 - 2.2. Ja statuss ir neaktīvs, tad sistēmai ir jāieslēdz autorizācijas lapu.
 - 2.3. Ja kāds no laukiem nav ievadīts, izvadīt par to paziņojumu.
3. Jānodrošina iespēja meklēt preces.
 - 3.1. Jāparedz iespēja meklēt preci pēc nosaukuma vai identifikācijas koda.
4. Jānodrošina iespēja filtrēt preces.
 - 4.1. Jāparedz iespēja filtrēt preces pēc dažādām kategorijām, minimālās cenas, maksimālās cenas, ražotāja, kategorijas.
 - 4.2. Jāparedz iespēja, ka katrs nākamais filtrs papildina iepriekšējā filtra spēku meklēt preci.

5. Jānodrošina administratoram iespēju pievienot preces.
 - 5.1. Jāparedz ieejas informācijas par preci (skat. 2.1.3. Ieejas informācijas apraksts) ievadīšana un pārbaude uz formāta pareizību.
 - 5.2. Ja kāds no obligātiem laukiem nav ievadīts, tad izvadīt par to kļūdas paziņojumu.
 - 5.3. Jāparedz iespēju izvadīt paziņojumu, ka prece ir veiksmīgi pievienota.
6. Jānodrošina administratoram iespēju rediģēt preces.
 - 6.1. Jāparedz ieejas informācijas par preci (skat. 2.1.3. Ieejas informācijas apraksts) ievadīšana un pārbaude uz formāta pareizību.
 - 6.2. Ja prece ir veiksmīgi rediģēta izvadīt paziņojumu par veiksmīgu rediģēšanu.
 - 6.3. Ja kāds no obligātiem laukiem nav ievadīts, tad izvadīt par to kļūdas paziņojumu.
 - 6.4. Ja kāds no obligātiem laukiem neatbilst validācijas prasībām, tad izvadīt par to kļūdas paziņojumu.
7. Jānodrošina administratoram iespēju dzēst preces.
 - 7.1. Pirms preces dzēšanas ir jāizvada paziņojums, vai tiešām vēlaties dzēst konkrēto ierakstu.
8. Jānodrošina iespēju administratoram pievienot kategorijas.
 - 8.1. Jāparedz ieejas informācijas par preci (skat. 2.1.2. Ieejas informācijas apraksts) ievadīšana un pārbaude uz formāta pareizību.
 - 8.2. Ja kāds no obligātiem laukiem nav ievadīts, tad izvadīt par to kļūdas paziņojumu.
 - 8.3. Jāparedz iespēju izvadīt paziņojumu, kad kategorija ir veiksmīgi pievienota.
9. Jānodrošina iespēju administratoram rediģēt kategorijas.
 - 9.1. Jāparedz ieejas informācijas par kategoriju (skat. 2.1.2. Ieejas informācijas apraksts) ievadīšana un pārbaude uz formāta pareizību.
 - 9.2. Ja kategorija ir veiksmīgi rediģēta izvadīt paziņojumu par veiksmīgu rediģēšanu.
 - 9.3. Ja kāds no obligātiem laukiem nav ievadīts, tad izvadīt par to kļūdas paziņojumu.
 - 9.4. Ja kāds no obligātiem laukiem neatbilst validācijas prasībām, tad izvadīt par to kļūdas paziņojumu.
10. Jānodrošina iespēju administratoram dzēst kategorijas.
 - 10.1. Pirms kategorijas dzēšanas ir jāizvada paziņojums, vai tiešām vēlaties dzēst konkrēto ierakstu.
11. Jānodrošina administratoram pievienot ražotājus.
 - 11.1. Jāparedz ieejas informācijas par preci (skat. 2.1.6. Ieejas informācijas apraksts) ievadīšana un pārbaude uz formāta pareizību.
 - 11.2. Jāparedz iespēju izvadīt paziņojumu, kad ražotājs ir veiksmīgi pievienots.
 - 11.3. Ja kāds no obligātiem laukiem nav ievadīts, tad izvadīt par to kļūdas paziņojumu.

12. Jānodrošina iespēju administratoram rediģēt ražotājus.
 - 12.1. Jāparedz ieejas informācijas par ražotāju (skat. 2.1.6. Ieejas informācijas apraksts) ievadīšana un pārbaude uz formāta pareizību.
 - 12.2. Ja ražotājs ir veiksmīgi rediģēts izvadīt paziņojumu par veiksmīgu rediģēšanu.
 - 12.3. Ja kāds no obligātiem laukiem nav ievadīts, tad izvadīt par to kļūdas paziņojumu.
 - 12.4. Ja kāds no obligātiem laukiem neatbilst validācijas prasībām, tad izvadīt par to kļūdas paziņojumu.
13. Jānodrošina iespēju administratoram dzēst ražotājus.
 - 13.1. Pirms ražotāja dzēšanas ir jāizvada paziņojums, vai tiešām vēlaties dzēst konkrēto ierakstu.
14. Jānodrošina iespēju reģistrētam lietotājam pievienot, noņemt preces no groza.
 - 14.1. Jānodrošina iespēja lietotājam pievienot vai noņemt preču daudzumu.
15. Jānodrošina iespēju reģistrētam lietotājam veikt pasūtījumu.
 - 15.1. Jānodrošina lietotājam pirms pasūtījuma veikšanas iespēju apskatīties, ko viņš pasūta, kādā daudzumā.
 - 15.2. Jānodrošina lietotājam iespēju izmainīt pasūtījuma detaļas, pirms pasūtījuma veikšanas.
16. Jānodrošina reģistrētam lietotājam apskatīt savu pasūtījumu vēsturi.
 - 16.1. Jānodrošina sadaļa, kurā lietotājs var pārskatīt visus veiktos pasūtījumus, to statusus un detaļas.
 - 16.2. Lietotājam jānodrošina iespēja filtrēt pasūtījumu vēsturi pēc datuma, statusa vai citiem kritērijiem.
17. Jānodrošina lietotājam veikt preču salīdzināšanu pēc specifikācijas.
18. Jānodrošina administratoram iespēju pabeigt vai atcelt pasūtījumus.
19. Jānodrošina administratoram iespēju eksportēt informāciju uz Excel formātu par vadības sistēmu.
 - 19.1. Jānodrošina iespēju eksportētu pilnīgi visi ieraksti ar vienu ar vienas pogas klikšķi.
 - 19.2. Jānodrošina iespēja eksportēt tikai nepieciešamie ieraksti.
20. Jānodrošina iespēja meklēt ierakstus administratora sistēmā ievadot nosaukumu vai ID.
21. Jānodrošina iespēja administratoram meklēt un filtrēt pirkumus.
 - 21.1. Jānodrošina iespēja meklēt pirkumus pēc telefona numura.
 - 21.2. Jānodrošina iespēja meklēt pirkumus pēc e-pasta.
 - 21.3. Jānodrošina iespēja filtrēt pirkumus pēc to statusa.
22. Jānodrošina iespēju lietotājiem iziet no portāla.

2.3. Nefunkcionālās prasības

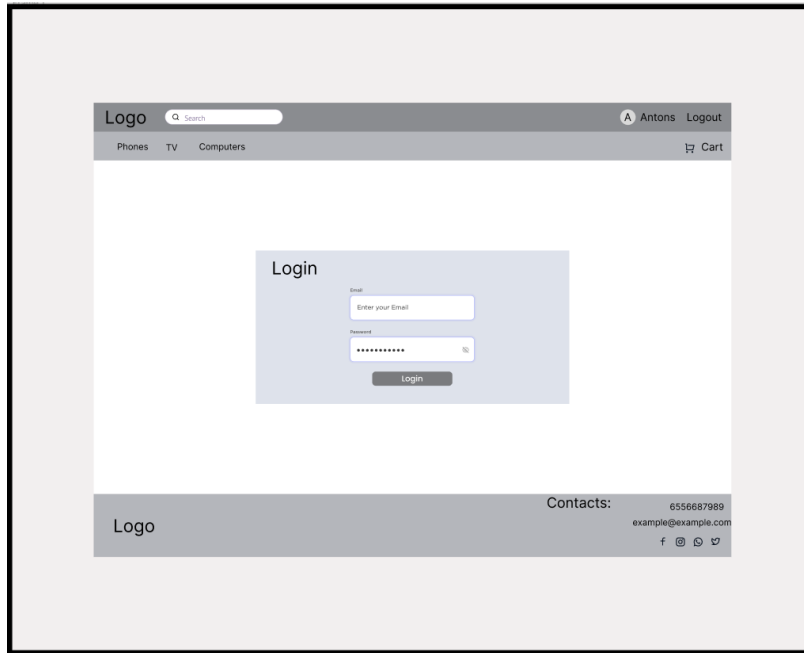
1. Sistēmas saskarnes valodai ir jābūt angļu valodai.
2. Jānodrošina tīmekļa lietojumprogrammas pielāgošanas ekrāna izmēriem, kas mūsdienās tiek lietoti, lai to varētu izmantot uz dažādiem monitora izmēriem.
3. Sistēmas dizainam ir jābūt nepārprotamam.
4. Tekstam jābūt tumšā krāsā ar viegli salasāmu fontu.
5. Sistēmai ir jāiekļauj uzņēmuma krāsu palete.
6. Sistēmai ir jābūt ātrdarbīgai.
7. Sistēma ir jābūt saderīgai ar populārākajiem tīmekļa pārlūkiem, Chrome, Firefox, Edge.
 - Sistēmas galvenās lapas skice (skat. 2.1. att.).



2.1. att. Sistēmas galvenās lapas skice

Šī skice attēlo sistēmas interfeisu, kuru redzēs pilnīgi visi, gan viesi, gan lietotāji, gan administratori. Skatā var redzēt reklāmas bannerus, pārdotākās preces, uzņēmuma logo, uzņēmuma kontaktinformāciju, preču sadaļas, kā arī iespēju iziet no sesijas un apskatīt savu profilu un grozu.

- Ienākšanas loga skice (skat. 2.2. att.)



2.2. att. Sistēmas galvenās lapas skice

Šī skice attēlo sistēmas interfeisu, kuru redzēs pilnīgi visi, gan viesi, gan lietotāji, gan administratori. Skatā var redzēt ienākšanas logu ar laukiem priekš paroles un e-pasta.

3. UZDEVUMA RISINĀŠANAS LĪDZEKĻU IZVĒLES PAMATOJUMS

Elektropreču internetveikala datu uzskaites sistēma ir paredzētā izmantošanai pārlūkprogrammās, gan datoros, gan mobilajās ierīcēs. Sistēma ir sadalīta divās daļās – lietotāja daļa (frontend) un servera puse (backend). Visi dati kurus ir nepieciešams uzglabāt un organizēt tiks glabāti datubāzē.

Lietotāja daļai (frontend) tiek izmantots **Vue.js** versija - 3.3.4 un **Vite.js** versija - 4.4.0. Vue.js un Vite.js kombinācija tiek izmantota, jo tā piedāvā ātru un efektīvu veidu, kā izstrādāt dinamiskas lietotāja saskarnes. Vue.js ir viegli uzturams un nodrošina modulāru komponentu struktūru. Vite.js nodrošina ātru izstrādi, uzlabojot attīstības laiku. Papildus tiek izmantots **Bootstrap** – versija 5.3, lai vieglāk būtu veidot konsekventus un responsīvus dizainus. Tiek izmantots **chart.js** lai izveidotu grafikus, **jspdf** lai izveidotu PDF izdrukas un **xlsx** lai izveidotu Excel izdrukas. Lai iegūtus datus no (backend) projektā tiek izmantots **Axios** versija - 1.6.2 tādējādi ļaujot lietotājam veikt HTTP pieprasījumus uz vai no servera izmantojot API.

Servera pusei (backend) tiek izmantots **XAMPP** Control Panel versija - 3.3.0, lai varētu palaist **Apache** versija - 2.4.54 un **MySQL**, versija - 10.4.27- **MariaDB**. XAMPP tiek izmantots, jo tas ļauj viegli izveidot lokālu serveri. Datubāzes izstrādei un uzturēšanai tiek izmantots **phpMyAdmin**, jo tas ir paredzēts MySQL datubāžu pārvaldīšanai izmantojot ērtu tīmekļa saskarni. Servera puses kodam (backend) tiek izmantots **PHP** versija - 8.0.26 un **Laravel** framework versija - 9.52.16, nodrošinot efektīvu un modernu veidu, kā strukturēt (backend) kodu, piedāvājot MVC (Model-View-Controller) arhitektūru.

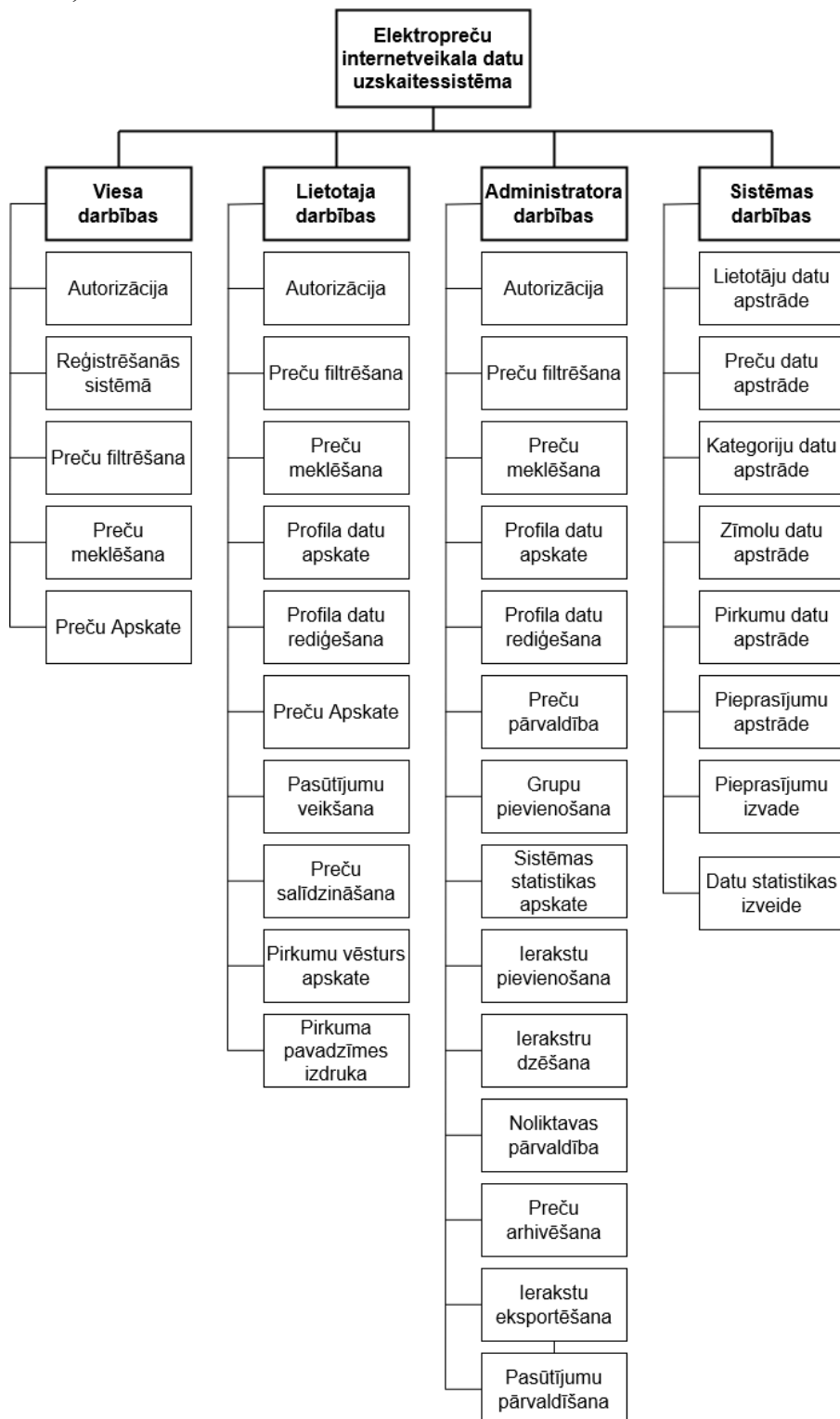
Sistēmas izstrādei ir izmantots **Visual Studio Code** versija - 1.85.0. kas ir atvērtā koda izstrādes vide. **Node.js** versija -18.13.0. kas ir atvērtā pirmkoda, starpplatformu JavaScript reāllaika vide, kas ļauj izpildīt JavaScript kodu servera pusē. **Git** versija - 2.32.0.windows.2. kas ir versiju kontroles sistēma ar uzsvaru uz ātrdarbību.

4. PROGRAMMATŪRAS PRODUKTA MODELĒŠANA UN PROJEKTĒŠANA

4.1. Sistēmas struktūras modelis

4.1.1. Sistēmas arhitektūra

Sistēma tiks veidota no četriem modeļiem (skat. 4.1. att.) – viesis, lietotājs, administrators, sistēma.

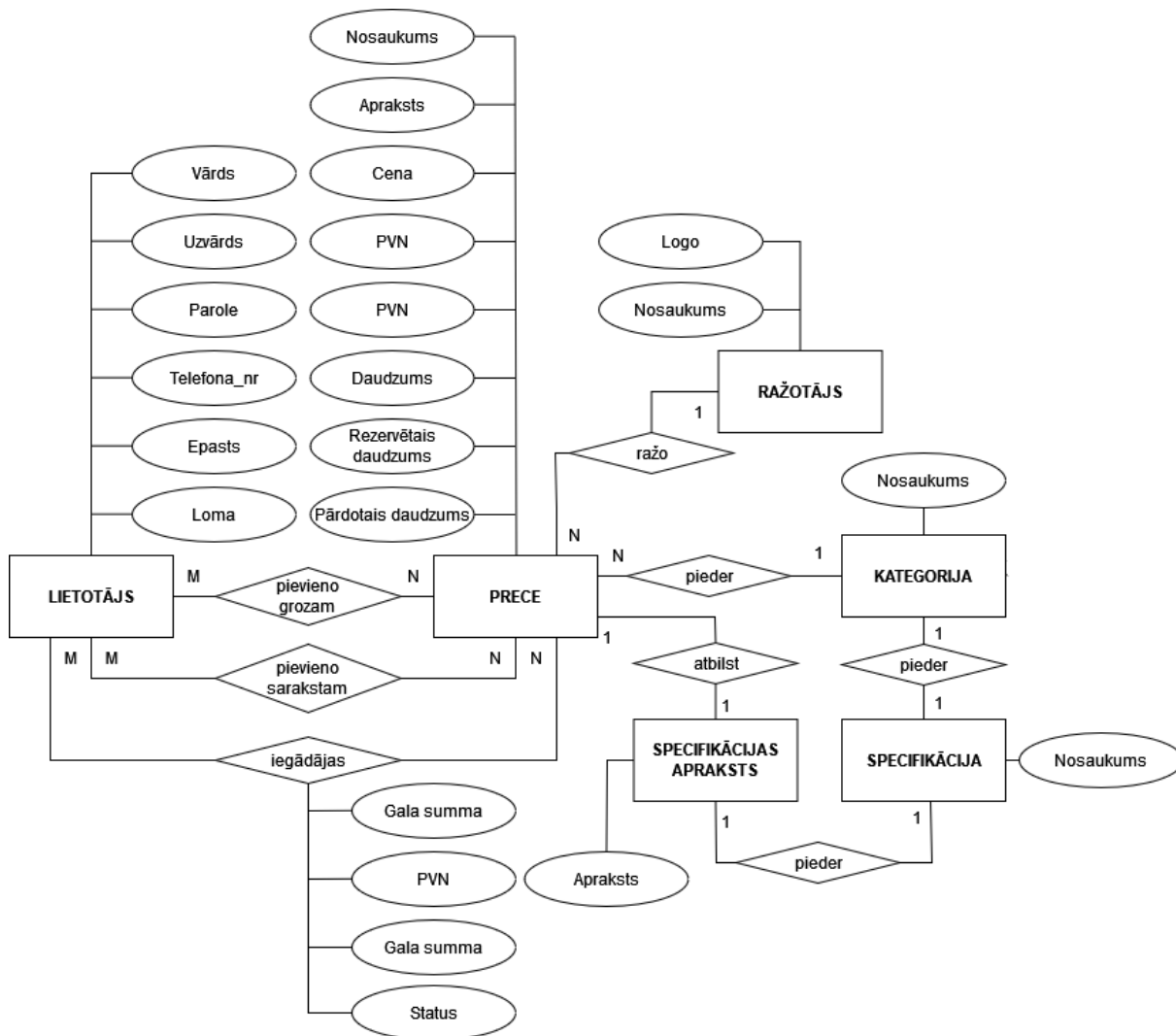


4.1. att. Funkcionālās dekompozīcijas diagramma

- **Viesu modelis.** Neregistrētam lietotājam – viesim būs iespēja izveidot jaunus sistēmas lietotājus, kā arī apskatīt preces, kuras atrodamas internetveikalā, tās meklēt un filtrēt. Viesiem pēc jauna sistēmas lietotāja izveides būs iespēja autorizēties, tādējādi kļūstot par sistēmas lietotāju.
- **Lietotāju modulis.** Reģistrētam lietotājam būs iespēja autorizēties sistēmā. Autorizējoties sistēmā lietotājam tiks dotas papildus funkcijas, kas nav pieejamas viesim. Lietotājam būs iespēja apskatīt preces, meklēt preces, filtrēt preces, salīdzināt preces un veidot savu pirkuma grozu no izvēlētajām precēm. Lietotājam būs iespēja apskatīt savus profila datus, kā arī rediģēt tos, ja ir nepieciešams, kā arī lietotājam būs iespēja veikt pasūtījumu no izvēlētajām precēm, un apskatīt pasūtījumu statusus un pasūtījumu vēsturi.
- **Administrators modelis.** Administratoram būs visas tās pašas funkcijas, kādas būs lietotājam, bet papildus administratoram būs iespēja pārvaldīt pasūtījumus, pievienot preces un to specifikāciju, pievienot kategorijas, zīmolus, kā arī visu to būs iespējams rediģēt. Būs iespējams nepieciešamības gadījumā dzēst ierakstus, ja sistēmas validācija to atļaus.
- **Sistēmas modelis.** Sistēmas modelis būs atbildīgs par datu pieprasījumu izvadi – vizuālā daļa, ko redzēs lietotāji, viesi un administratori, kā arī datu pieprasījumu apstrādi. Kā arī sistēma atbildēs par visu datu apstrādi, to glabāšanu dzēšanu, rediģēšanu, statistikas, PDF, Excel izveidēm.

4.1.2. Sistēmas ER-modelis

Sistēmas ER-modelis sastāv no 6 entītijām (skat. 4.2. att.), kas nodrošina pamat informācijas uzglabāšanu un apstrādi. Tie ir Lietotājs, Prece, Preces kategorija, Grozs, Pasūtījums, Zīmoli.



4.2.att. Sistēmas ER-diagramma

- **Lietotājs** – apraksta lietotāju. Atribūtu kopums sevī ietver lietotajā vārdu, uzvārdu, paroli, telefona numuru, e-pastu, lomu.
- **Prece** – apraksta preci. Atribūtu kopums sevī ietver preces nosaukumu, preces aprakstu, preces cena, preces pievienotās vērtības nodoklis, preces daudzums, preces rezervētais daudzums, preces pārdotais daudzums..
- **Kategorija** - apraksta preces kategorija. Atribūtu kopums sevī ietver preces kategorijas nosaukumu.
- **Ražotājs** – apraksta preces ražotājus. Atribūtu kopums sevī ietver preces ražotāja nosaukumu un ražotāja logotipa attēlu.

- **Specifikācija** – apraksta kādas specifikācijas var būt kategorijai. Atribūtu kopums sevī ietver specifikācijas nosaukumu.
- **Specifikācijas apraksts** – apraksta preces specifikāciju attiecīgi pēc kategorijas. Atribūtu kopums sevī ietver aprakstu.

Datu bāzes relācijas uzrāda kā savstarpēji ir savienotas divas entitijas.

- Starp **Prece** un **Kategorija** attiecība ir viens pret daudziem, jo vienai kategorijai var būt vairākas preces, bet viena prece var piederēt tikai pie vienas preces kategorijas.
- Starp **Prece** un **Ražotājs** attiecība ir viens pret daudziem, jo vienam ražotājam var būt vairākas preces, bet vienai precei var būt tikai viens zīmols.
- Starp **Lietotājs** un **Prece** attiecība daudz pret daudziem, jo vienam lietotājam var būt vairākas preces grozā un vairākas preces var būt vairāku lietotāju grozos.
- Starp **Kategorija** un **Specifikācija** attiecība ir viens pret viens, jo vienai kategorijai var būt viena specifikācija un viena specifikācija var piederēt tikai vienai kategorijai.
- Starp **Prece** un **Specifikācijas apraksts** attiecība ir viens pret vienu, jo vienam vienai precei var būt viens specifikācijas apraksts un vienam specifikācijas aprakstam var būt vien prece.
- Starp **Specifikācijas apraksts** un **Specifikācija attiecība** ir viens pret vienu, jo vienai specifikācijai var piederēt tikai viens specifikācijas apraksts un vienam specifikācijas aprakstam var piederēt tikai viena specifikācija

4.2. Funkcionālais sistēmas modelis

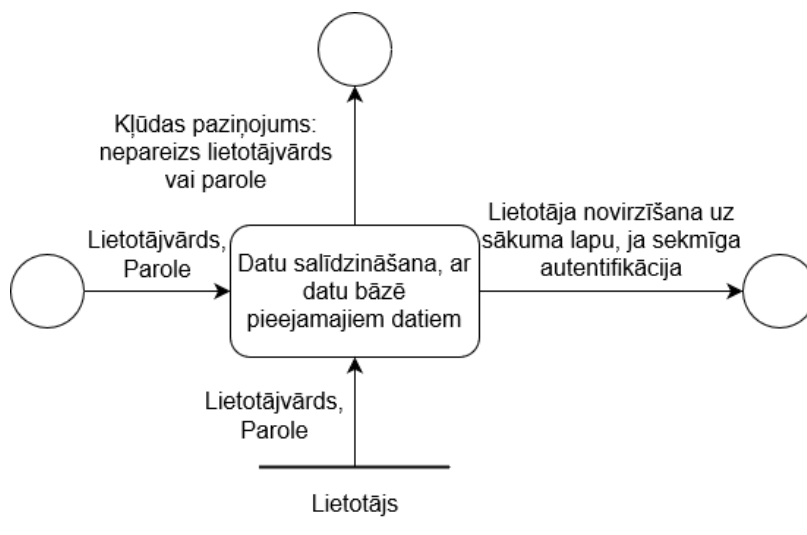
4.2.1. Datu plūsmu modelis

1. Lietotāja profila datu rediģēšana

Lietotāja profila datu rediģēšana atbildēs par konkrēta lietotāja datu rediģēšanu. Lietotāju datu rediģēšana sadarbojas ar profila datu izvadi. Ja lietotājs pamana, ka dati ir nekorekti vai mainījušies, tam būs iespēja rediģēt šos datus. Ja datu validācija būs sekmīga, tad lietotāja dati tiks rediģēti datu bāzē. Ja validācija būs neveiksmīga, tad tiks izvadīts kļūdas paziņojums – kura no ailēm neatbilst prasībām.

2. Lietotāju autorizēšanās

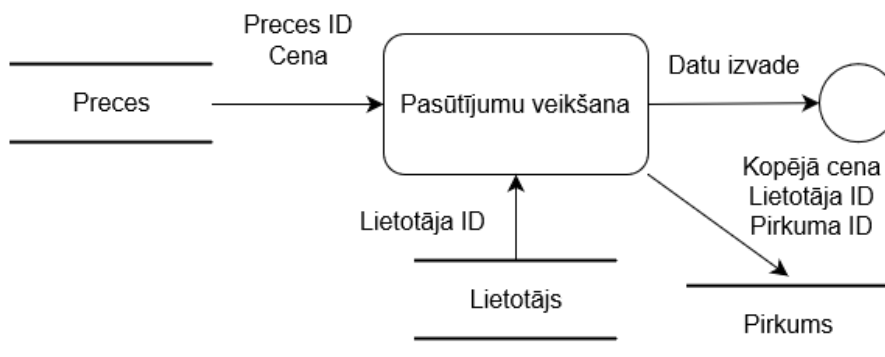
Autorizācijas modelis (skat. 4.3. att.) atbildēs par lietotāja autorizāciju sistēmā. Lietotājam ir jāievada e-pasts un parole. Ja lietotājs ir ievadījis pareizus datus, tad tiks ielaists sistēmā, kā konkrētais lietotājs. Ja dati būs ievadīti nekorekti, tad tiks izvadīts paziņojums, ka e-pasts vai parole ir nepareizs.



4.3. att. Lietotāja autorizācijas datu plūsmu diagramma

3. Lietotāja reģistrēšana datu bāzē

Reģistrēšanas modelis (skat. 4.4. att.) atbildēs par lietotāja reģistrēšanu datu bāzē. Viesim būs jāizpilda ailes, kurās jāievada e-pasts, vārds, uzvārds, parole, parole atkārtoti un telefona numurs. Ja visi lauki būs aizpildīti un tie atbildīs visām validācijas prasībām, tad tiks izveidots jauns lietotāja profils un būs iespēja reģistrēties sistēmā. Ja kāda no validācijas prasībām nebūs veiksmīga, tad viesim tiks izvadīts kļūdas paziņojums – kura no ailēm neatbilst prasībām.



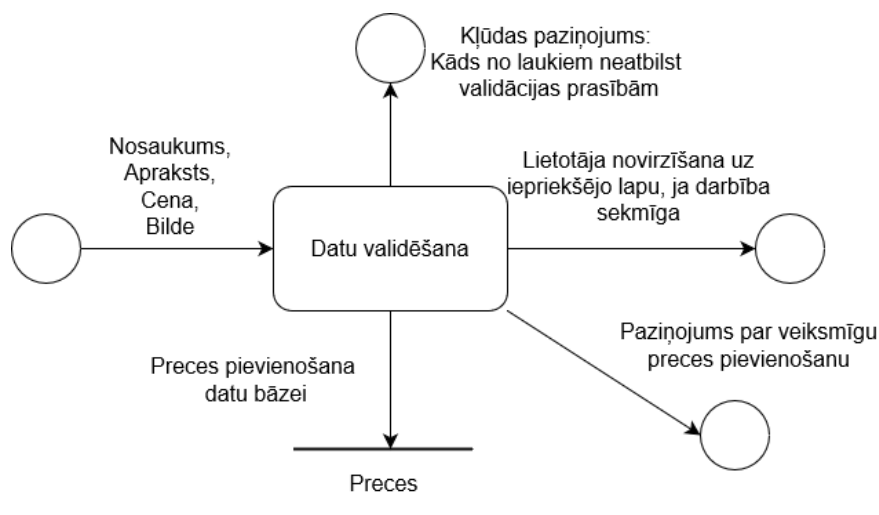
4.4. att. Lietotāja reģistrēšanas datu plūsmu diagramma

4. Profila datu izvade

Profilu datu izvade atbildēs par konkrētā reģistrētā lietotāja datu izvadi, šos datus redzēs tikai un vienīgi profila īpašnieks. Profila īpašniekam par sevi būs iespējams aplūkot šādus datus – vārds, uzvārds, e-pasts, telefona numurs.

5. Preču ierakstu pievienošana datu bāzē

Preču ierakstu pievienošana (skat. 4.5. att.) atbildēs par preces pievienošanu datu bāzei. Administratoram būs iespēja pievienot jaunu preci sistēmai aizpildot ievades laukus. Lai pievienotu jaunu preci ir nepieciešams ievadīt preces nosaukumu, aprakstu, cenu, attēlu. Ja datu validācija būs sekmīga, tad prece tiks pievienota datu bāzei. Ja kāda no validācijas prasībām nebūs veiksmīga, tad tiks izvadīts kļūdas paziņojums – kura no ailēm neatbilst prasībām un prece netiks pievienota datu bāzei.



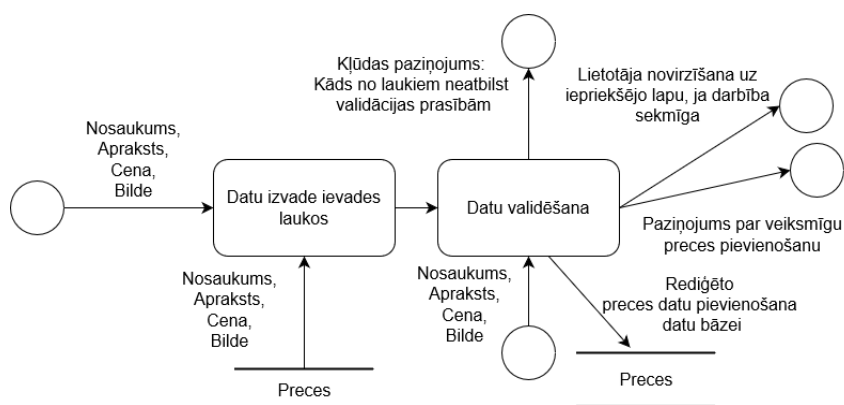
4.5. att. Preču pievienošanas datu plūsmu diagramma

6. Ierakstu meklēšana

Ierakstu meklēšana atbildēs par preču meklēšanu pēc specifiskas frāzes vai vārda vai burtiem. Speciālā meklēšanas laukā lietotājam būs jāievada piemēram konkrēts preces nosaukums, pēc tam pēc šīs frāzes tiks meklēts, vai atbilst kāds ieraksts, ja ieraksts datubāzē tiks atrasts, tad tas tiks izvadīts, ja ieraksts netiks atrasts tiks izvadīts paziņojums, ka neviena prece neatbilst meklētajai.

7. Preču ierakstu rediģēšana datu bāzē

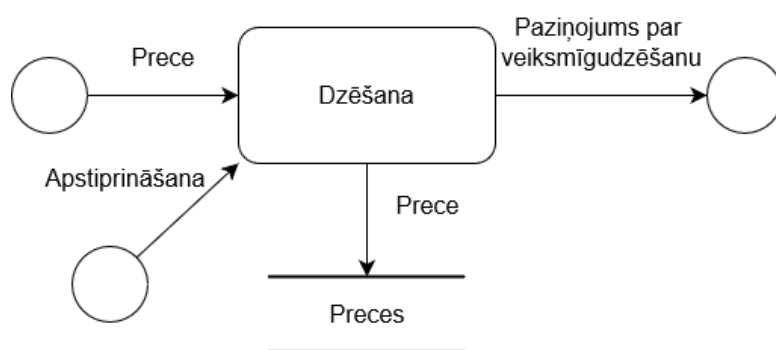
Preču datu rediģēšana (skat. 4.6. att.) atbildēs par konkrētas izvēlētās preces datu rediģēšanu. Administratoram būs nepieciešams attiecīgajās ailēs, kurās tas būs nepieciešams izmainīt datus piemēram cenu. Pēc datu rediģēšanas dati izies cauri validācijai, ja viss atbildīs prasībām, tad dati tiks veiksmīgi rediģēti datubāzē, kā arī datubāzē uzrādīsies, kad dati ir rediģēti. Ja validācija būs neveiksmīga, tad tiks izvadīts kļūdas paziņojums – kura no ailēm neatbilst prasībām.



4.6. att. Preču rediģēšanas datu plūsmu diagramma

8. Preču ierakstu dzēšana datu bāzē

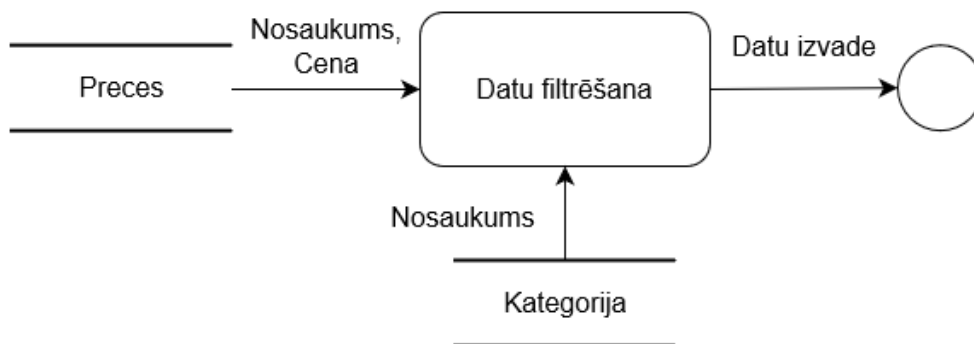
Preču ierakstu dzēšana (skat. 4.7. att.) atbildēs par konkrētas izvēlētās preces dzēšanu no datu bāzes ierakstiem. Administratoram būs jāizvēlas, kuru preci vēlas dzēs, pēc tā tiks izvadīts paziņojums, vai tiešām vēlaties dzēst preci, ja tiks nospiests jā, tad ieraksts tiks dzēsts no datu bāzes, ja lapa tiks aizvērta vai uzspiests nē, tad nekas nenotiks.



4.7. att. Preču dzēšanas datu plūsmu diagramma

9. Ierakstu filtrēšana

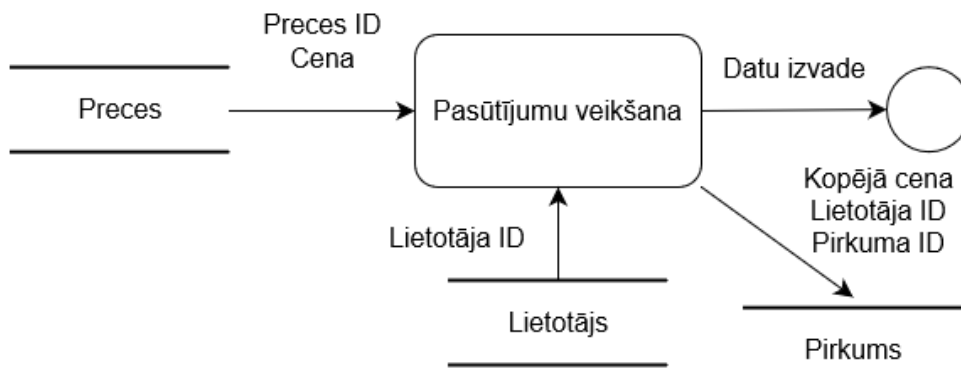
Ierakstu filtrēšana (skat. 4.8. att.) atbildēs par preču filtrēšanu, būs iespēja izvēlēties, kā kārot preces piemēram cena augoša vai cena dilstoša, cena no līdz. Lietotājam ievadot kādu no filtrēšanas opcijām dati tiks attiecīgi filtrēti un izvadīti tikai filtriem atbilstošie dati. Ja iestatot filtrus neviena prece neatbildīs prasībām, tad tiks izvadīts neviena prece netika atrasta.



4.8. att. Ierakstu filtrēšanas datu plūsmu diagramma

10. Pasūtījumu veikšana

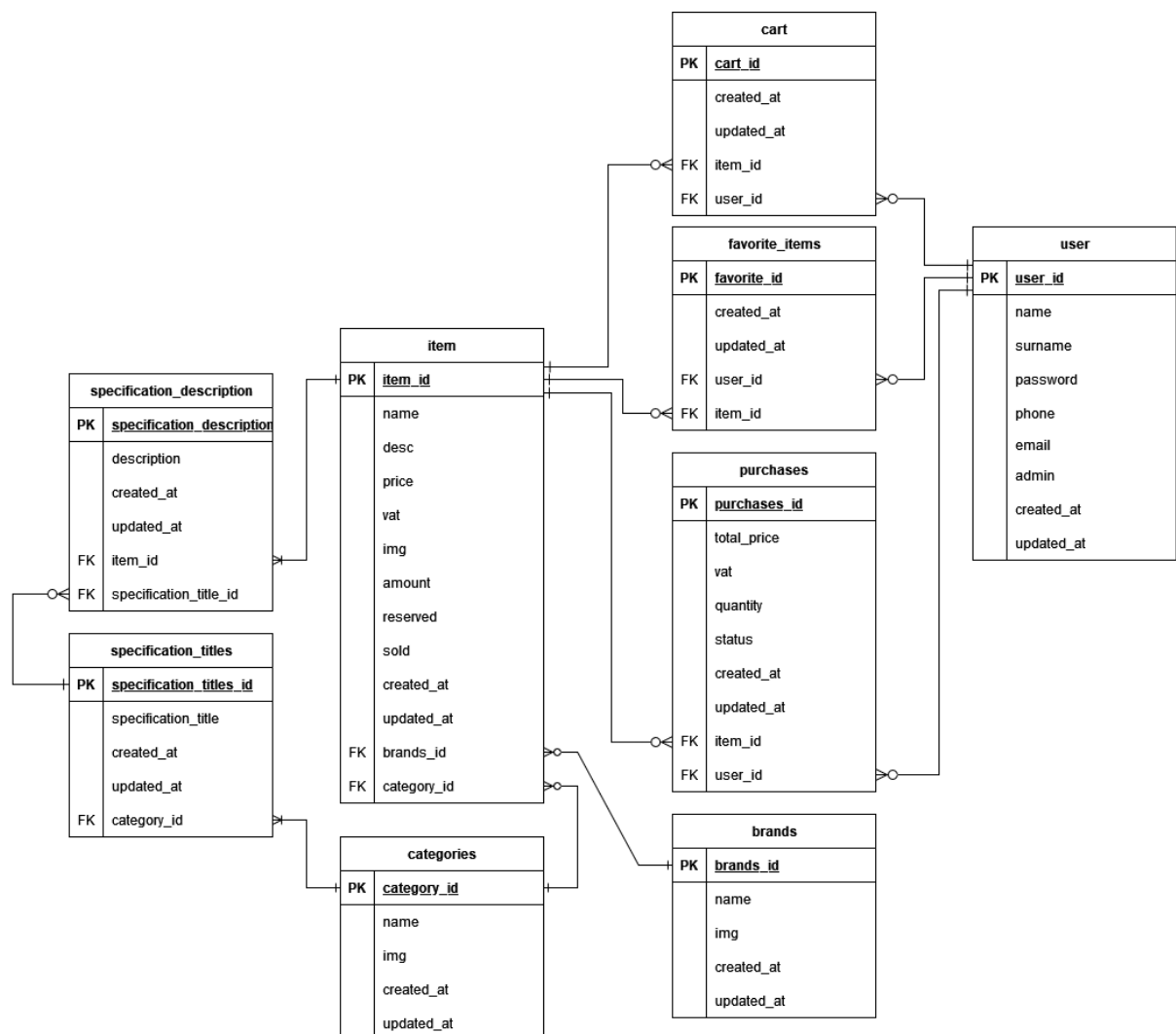
Pasūtījumu veikšana (skat. 4.9. att.) atbildēs par gala pasūtījuma veikšanu. Lietotājam veicot pasūtījumu tiks prasīts apstiprināt izvēlētās preces un to gala cenu, ja lietotājs izvēlas doties tālāk, tad tiks izveidots pasūtījums un lietotājam būs iespēja izrentēt pasūtījuma dokumentu. Ja lietotājs neizvēlas turpināt pasūtījuma, tad tas netiek izveidots un lietotājs tiek novirzīts atpakaļ uz iepriekšējo lapu.



4.9. att. Pasūtījumu veikšanas datu plūsmu diagramma

5. DATU STRUKTŪRU APRAKSTS

Datubāze sastāv no 6 tabulām (skat. 5.1. att.), kas satur sevī informāciju par lietotājiem, precēm, kategorijām, grozu, pasūtījumu un groza objektiem.



5.1. att. Tabulu relāciju shēma

1. Tabula “**user**” glabā datus par lietotājiem.
2. Tabula “**item**” glabā datus par precēm.
3. Tabula “**categories**” glabā datus par kategorijām.
4. Tabula “**purchases**” glabā datus par pasūtījumiem.
5. Tabula “**cart**” glabā datus par precēm, kurus lietotājs ielicis grozā.
6. Tabula “**brands**” glabā datus par zīmoliem.
7. Tabula “**favorite_items**” glabā datus par iecienītākajām lietotāja precēm.
8. Tabula “**specification_description**” glabā datus par specifikāciju aprakstiem.
9. Tabula “**specification_titles**” glabā datus par specifikāciju virsrakstiem.

Tabula “**user**” glabā datus par reģistrētiem lietotājiem. Tabulas “**user**” primārā atslēga ir “**user_id**”, tabula ir saistīta ar tabulu “**cart**”.

5.1. tabula

Tabulas “**user**” struktūra

Nr.	Nosaukums	Tips	Izmērs	Apraksts
1.	user_id	bigint	20	Ieraksta identifikators, primārā atslēga (unikāls)
2.	name	varchar	255	Lietotāja vārds
3.	surname	varchar	255	Lietotāja uzvārds
4.	password	varchar	255	Šifrēta parole, ko lietotājs ir izvēlējis
5.	phone	int	11	Lietotāja telefona numurs
6.	email	varchar	255	Lietotāja e-pasts
7.	admin	tinyint	1	Lietotāja loma
8.	created_at	timestamp	-	Kad tika izveidots lietotājs
9.	updated_at	timestamp	-	Kad tika atjaunoti lietotāja dati

Tabula “**favorite_items**” glabā datus par iecienītākajām lietotāja precēm, tabulai ir unikāla primārā atslēga “**category_id**”. Tabula “**categories**” ir saistīta ar tabulām “**item**” un “**users**”, tabulai ir arī atslēga “**user_id**”, kas atsaucas uz tabulu “**users**” un “**item_id**”, kas atsaucas uz tabulu “**item**”.

5.2. tabula

Tabulas “**favorite_items**” struktūra

Nr.	Nosaukums	Tips	Izmērs	Apraksts
1.	favorite_id	bigint	20	Ieraksta identifikators, primārā atslēga (unikāls)
2.	item_id	bigint	20	Ārējā atslēga
3.	user_id	bigint	20	Ārējā atslēga
4.	created_at	timestamp	-	Kad tika izveidots lietotājs
5.	updated_at	timestamp	-	Kad tika atjaunoti lietotāja dati

Tabula “**categories**” glabā datus par preču kategorijām, tabulai ir unikāla primārā atslēga “category_id”. Tabula “**categories**” ir saistīta ar tabulu “**item**”.

5.3. tabula

Tabulas “**categories**” struktūra

Nr.	Nosaukums	Tips	Izmērs	Apraksts
1.	category_id	bigint	20	Ieraksta identifikators, primārā atslēga (unikāls)
2.	category_name	varchar	255	Kategorijas nosaukums
4.	created_at	timestamp	-	Kad tika izveidots lietotājs
5.	updated_at	timestamp	-	Kad tika atjaunoti lietotāja dati

Tabula “**item**” glabā datus par precēm, tabula ir saistīta ar trim citām “**categories**”, “**cart**” un “**brands**”. Tabulas “**item**” primārā atslēga ir “item_id”, kas apraksta katras preces unikālo identifikatoru, kā arī tabulai ir ārējās atslēgas “category_id” un “brands_id”, kas atsaucas uz tabulām “**categories**” un “**brands**”.

5.4. tabula

Tabulas “**item**” struktūra

Nr.	Nosaukums	Tips	Izmērs	Apraksts
1.	item_id	bigint	20	Ieraksta identifikators, primārā atslēga (unikāls)
2.	name	varchar	255	Preces nosaukums
3.	desc	varchar	255	Preces apraksts
4.	price	float	-	Preces cena
5.	img	varchar	255	Tiek glabāts preces attēla ceļš -EVk72G.png
6.	vat	int	11	Tiek glabāts pvn apmērs
7.	amount	int	11	Tiek glabāt cik preces ir iespējams nopirkt
8.	rederved	int	11	Tiek glabāts cik preces ir rezervētas
9.	sold	int	11	Tiek glabāts cik preces ir pārdotas
10.	brands_id	bigint	20	Ārējā atslēga
11.	category_id	bigint	20	Ārējā atslēga
12.	created_at	timestamp	-	Kad tika izveidots lietotājs
13.	updated_at	timestamp	-	Kad tika atjaunoti lietotāja dati

Tabula “**cart**” glabā datus par lietotāja pirkuma grozu. Tabulai ir unikāla identifikācijas atslēga “basket_id”, kas arī ir primārā atslēga. Tabula “**cart**” ir saistīta ar tabulām “**item**” un “**users**”, tabulai ir ārējā atslēga “user_id”, kas atsaucas uz tabulu “**users**” un “item_id”, kas atsaucas uz tabulu “**item**”.

5.5. tabula

Tabulas “**cart**” struktūra

Nr.	Nosaukums	Tips	Izmērs	Apraksts
1.	cart_id	bigint	20	Ieraksta identifikators, primārā atslēga (unikāls)
2.	user_id	bigint	20	Ārējā atslēga
3.	Item_id	bigint	20	Ārējā atslēga
4.	created_at	timestamp	-	Kad tika izveidots lietotājs
5.	updated_at	timestamp	-	Kad tika atjaunoti lietotāja dati

Tabula “**purchases**” glabā datus par pasūtījumiem. Tabula ir saistīta ar tabulām “**item**” un “**users**”. Tabula “**purchases**” izmanto ārējās atslēgas “item_id” un “user_id”, lai izveidotu relācijas starp tabulām “**item**” un “**users**” tabulām.

5.6. tabula

Tabulas “**purchases**” struktūra

Nr.	Nosaukums	Tips	Izmērs	Apraksts
1.	purchases_id	bigint	20	Ieraksta identifikators, primārā atslēga (unikāls)
2.	status	varchar	255	Pasūtījuma status active vai closed vai canceled
3.	total_price	decimal	8,2	Kopējā cena, kas ir jāmaksā
4.	quantity	int	11	Cik preces vienības lietotājs iegādājās
5.	vat	int	11	Preces PVN daudzums
6.	item_id	bigint	20	Ārējā atslēga
7.	user_id	bigint	20	Ārējā atslēga
8.	created_at	timestamp	-	Kad tika izveidots lietotājs
9.	updated_at	timestamp	-	Kad tika atjaunoti lietotāja dati

Tabula “**brands**” glabā datus par zīmoliem. Tabulai ir unikāla identifikācijas atslēga “brands_id”, kas arī ir primārā atslēga. Tabula “**brands**” nav nevienas ārējās atslēgas.

5.7. tabula

Tabulas “**brands**” struktūra

Nr.	Nosaukums	Tips	Izmērs	Apraksts
1.	brands_id	bigint	20	Ieraksta identifikators, primārā atslēga (unikāls)
2.	name	varchar	255	Zīmola nosaukums
3.	img	varchar	255	Tiek glabāts preces attēla ceļš - EVkLOgI2G.png
4.	created_at	timestamp	-	Kad tika izveidots lietotājs
5.	updated_at	timestamp	-	Kad tika atjaunoti lietotāja dati

Tabula “**specification_description**” glabā datus par specifikāciju aprakstiem. Tabulai ir unikāla identifikācijas atslēga “specification_description_id”, kas arī ir primārā atslēga. Tabula “**specification_description**”. Tabula “**specification_description**” izmanto ārējās atslēgas “item_id” un “specification_title_id”, lai izveidotu relācijas starp tabulām “**item**” un “**specification_title**” tabulām.

5.8. tabula

Tabulas “**specification_description**” struktūra

Nr.	Nosaukums	Tips	Izmērs	Apraksts
1.	specification_description_id	bigint	20	Ieraksta identifikators, primārā atslēga (unikāls)
2.	description	varchar	255	Specifikācijas apraksts
3.	item_id	bigint	20	Ārējā atslēga
4.	specification_title_id	bigint	20	Ārējā atslēga
5.	created_at	timestamp	-	Kad tika izveidots lietotājs
6.	updated_at	timestamp	-	Kad tika atjaunoti lietotāja dati

Tabula “**specification_titles**” glabā datus par specifikāciju virsrakstiem. Tabulai ir unikāla identifikācijas atslēga “**specification_titles_id**”, kas arī ir primārā atslēga. Tabula “**specification_titles**”. Tabula “**specification_titles**” izmanto ārējās atslēgu “**category_id**”, lai izveidotu relācijas ar tabulu “**categories**”.

5.9. tabula

Tabulas “**specification_titles**” struktūra

Nr.	Nosaukums	Tips	Izmērs	Apraksts
1.	specification_titles_id	begin	20	Ieraksta identifikators, primārā atslēga (unikāls)
2.	specification_title	varchar	255	Specifikācijas virsraksts
3.	category_id	bigint	20	Ārējā atslēga
5.	created_at	timestamp	-	Kad tika izveidots lietotājs
6.	updated_at	timestamp	-	Kad tika atjaunoti lietotāja dati

Tabulu saišu attiecības:

- starp tabulu **specification_titles** un **specification_description** ir attiecība viens pret nulli vai vairākiem;
- starp tabulu **categories** un **specification_titles** ir attiecība viens pret vienu vai vairākiem;
- starp tabulu **categories** un **item** ir attiecība viens pret nulli vai vairākiem;
- starp tabulu **item** un **specification_description** ir attiecība viens pret vienu vai vairākiem;
- starp tabulu **brands** un **item** ir attiecību viens pret nulli vai vairākiem;
- starp tabulu **item** un **purchases** ir attiecība viens pret nulle vai vairākiem;
- starp tabulu **item** un **favorite_items** ir attiecība viens pret nulle vai vairākiem;
- starp tabulu **item** un **cart** ir attiecība viens pret nulle vai vairākiem;
- starp tabulu **user** un **purchases** ir attiecība viens pret nulle vai vairākiem;
- starp tabulu **user** un **favorite_items** ir attiecība viens pret nulle vai vairākiem;
- starp tabulu **user** un **cart** ir attiecība viens pret nulle vai vairākiem;

6. LIETOTĀJA CEĻVEDIS

6.1.Sistēmas prasības aparatūrai un programmatūrai

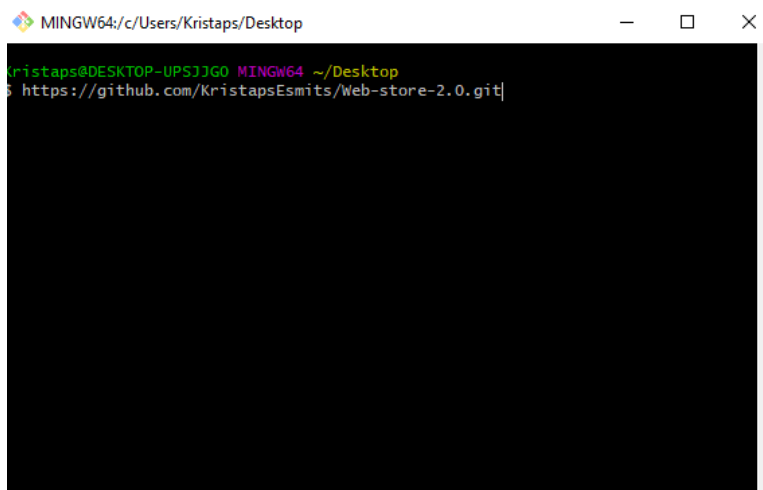
Sistēma tika realizēta kā interneta timekļa vietne, kuras lietošana neprasa specifisku programmu instalāciju, kā arī var tikt palaista no jebkuras ierīces, kurai ir:

- Viena no jaunākajām pārlūkprogrammu versijām
 - Safari – 13. versija un jaunāka;
 - Windows Edge – 100.0 un jaunāk;
 - Google Chrome – 100.0 un jaunāka;
 - Opera – 84.0.4316.31 un jaunāka;
 - Mozilla Firefox – 100.0 un jaunāka;
- Stabils interneta savienojums – jo labāks savienojums, jo veiksmīgāk noritēs programmas darbība;
- Aparatūra ar vismaz 2GB RAM, lai uz tās varētu darbināt iepriekš šajā sadaļā minētās pārlūkprogrammas.

No sistēmas uzturētāja puses ir nepieciešama aparatūra ar vismaz 2GB RAM un instalētām programmām – Visual studio code, XAMP, Git, nodeJS, kā arī kādai no iepriekš minētajām pārlūkprogrammām.

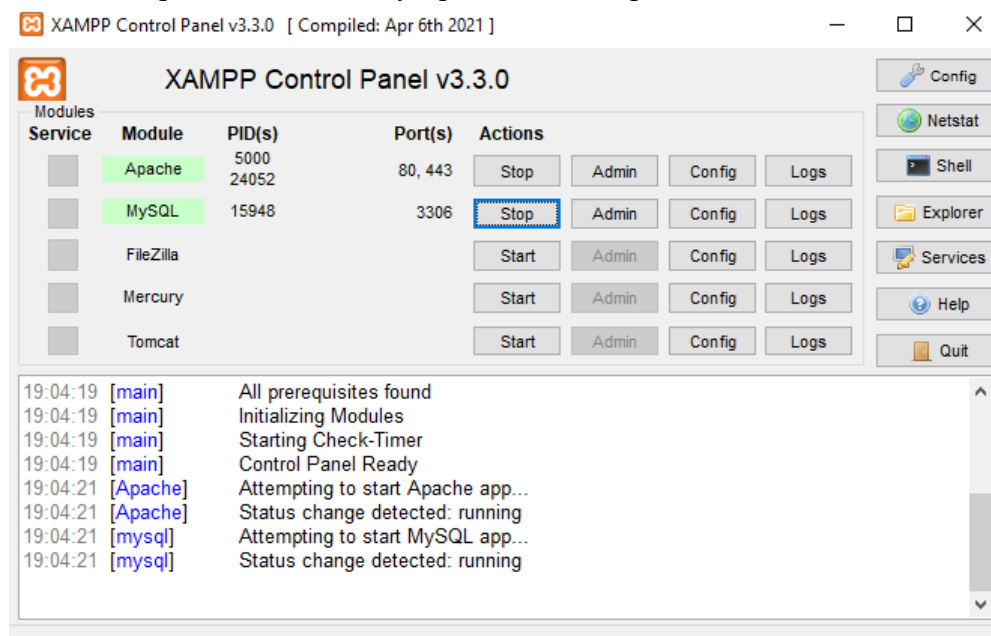
6.2.Sistēmas instalācija un palaišana

Sistēmas instalācijai un palaišanai būs nepieciešams papildus instalēt xamp, git. Nepieciešams instalēt Git, lai būtu iespējams noklonēt projektu no Github. Git var lejupielādēt <https://git-scm.com/downloads>. Jāizvēlas atbilstošā datora operētājsistēma un versija – portable. Ielādējot programmu un atverot to padīdīsies konsoles logs, kurā jāievada projekta github links – “<https://github.com/KristapsEsmits/Web-store-2.0.git>” (skat. 6.1. att.).



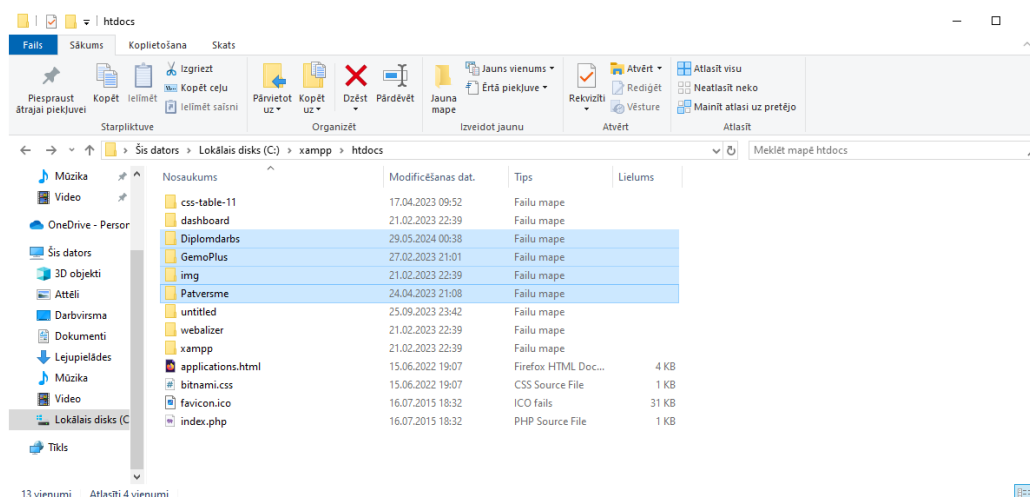
6.1. att. Git konsoles logs

Uzspiežot enter taustiņu tiks lejupielādēts projekts. Lai projektu palaistu ir nepieciešams instalēt XAMP^[9] – “<https://www.apachefriends.org/download.html>” lietotājs lejupielādē attiecīgi sev nepieciešamo versiju. Palaist programmu lietotājam jāpalaiž Apache server un MySql server (skat. 6.2. att.). Palaist Apache server un MySql serveri n nepieciešams katru reizi ieslēdzot sistēmu, Apache server un MySql server atbild par backend un datubāzes darbību



6.2. att. Xamp programmas logs

Tālāk jāpārkopē vai jāpārliet lejupielādētais Github projekts XAMP mapē - xampp\htdocs (skat. 6.3. att.), ja tas netiks izdarīts XAMP nespēs atrast projektu un datubāze nekomunicēs savā starpā ar backend un otrādi.

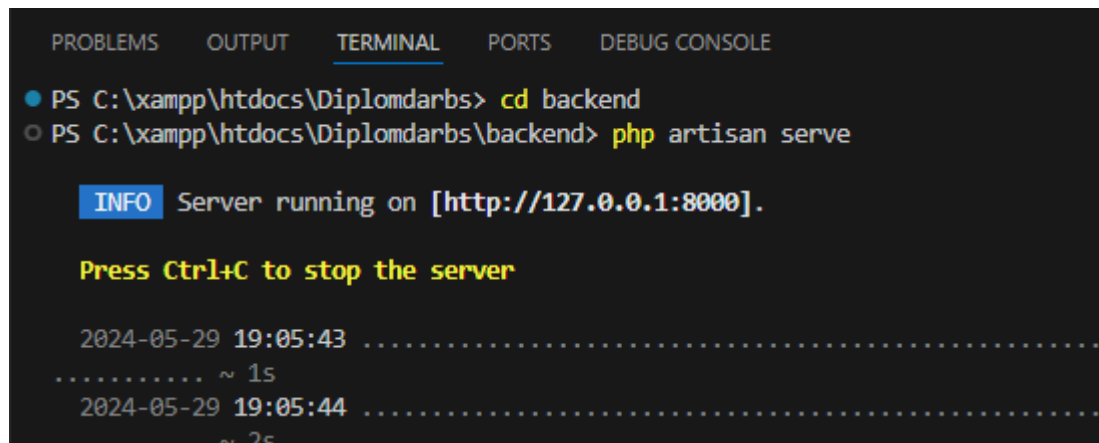


6.3. att. Piemērs, kur jāiekopē novilktais projekts

Tālāk jāatver projekts konsolē, vai koda redaktora konsolē un jāievada sekojošas komandas (skat. 6.4. att.).

* Ievadot šīs komandas tiek palaists projekta backend

- `cd backend`
- `php artisan serve`



```
PROBLEMS  OUTPUT  TERMINAL  PORTS  DEBUG CONSOLE
● PS C:\xampp\htdocs\Diplomdarbs> cd backend
○ PS C:\xampp\htdocs\Diplomdarbs\backend> php artisan serve

INFO Server running on [http://127.0.0.1:8000].

Press Ctrl+C to stop the server

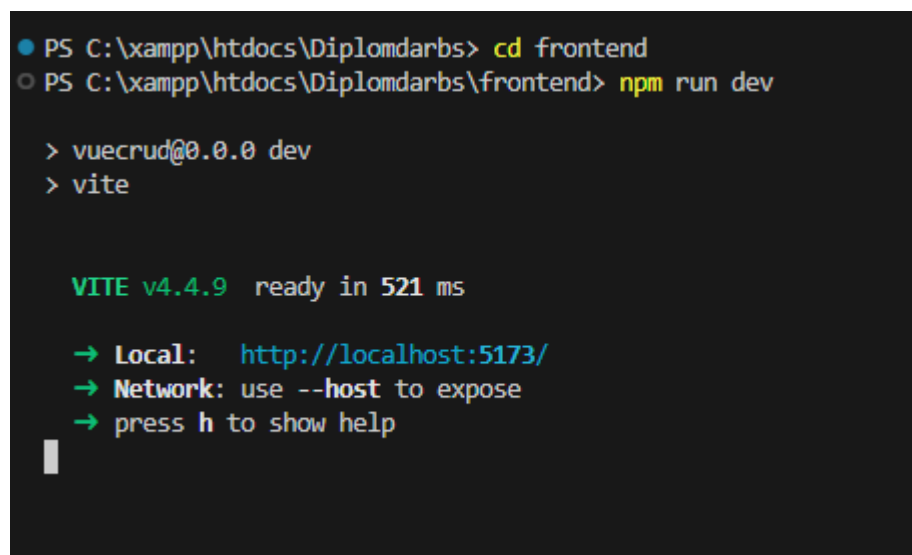
2024-05-29 19:05:43 .....
..... ~ 1s
2024-05-29 19:05:44 .....
..... ~ 2s
```

6.4. att. Konsole ar ievadītām komandām priekš backend palaišanas

Tālāk jāatver jauna konsole neaizverot ciet esošo konsoli un jāievada sekojošas komandas (skat. 6.5. att.):

* Ievadot šīs komandas tiek palaists projekta frontend

- `cd frontend`
- `npm run dev`



```
● PS C:\xampp\htdocs\Diplomdarbs> cd frontend
○ PS C:\xampp\htdocs\Diplomdarbs\frontend> npm run dev

> vuecrud@0.0.0 dev
> vite

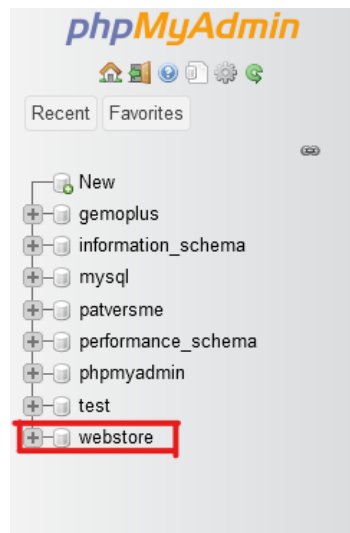
VITE v4.4.9 ready in 521 ms

→ Local:   http://localhost:5173/
→ Network: use --host to expose
→ press h to show help
```

6.5. att. Konsole ar ievadītām komandām priekš frontend palaišanas

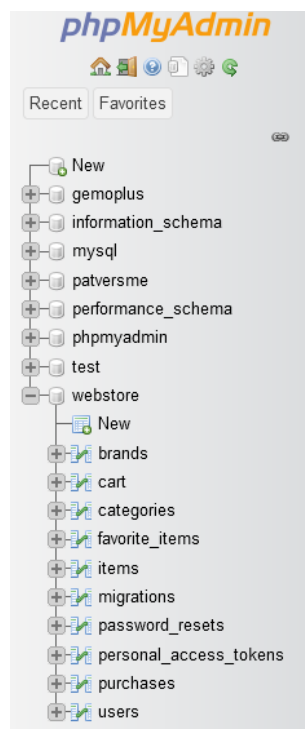
Sekmīgi veicot visas iepriekš minētās darbības jābūt iespējai atvērt projektu lokāli, to var izdarīt spiecot uz linka, kas tiek izvadīts konsolē (skat. 6.2.5. att.). Lai piekļūtu datubāzei vai veiktu manipulācijas ar to nepieciešams atvērt linku: <http://localhost/phpmyadmin/> un izveidot jaunu datubāzi ar nosaukumu webstore (skat. 6.6. att.). Tālāk jāatver projekts jaunā konsolē un jāievada sekojošas komandas:

- `cd backend`
- `php artisan migrate`



6.6. att. Attēls ar izveidotu jaunu datubāzi

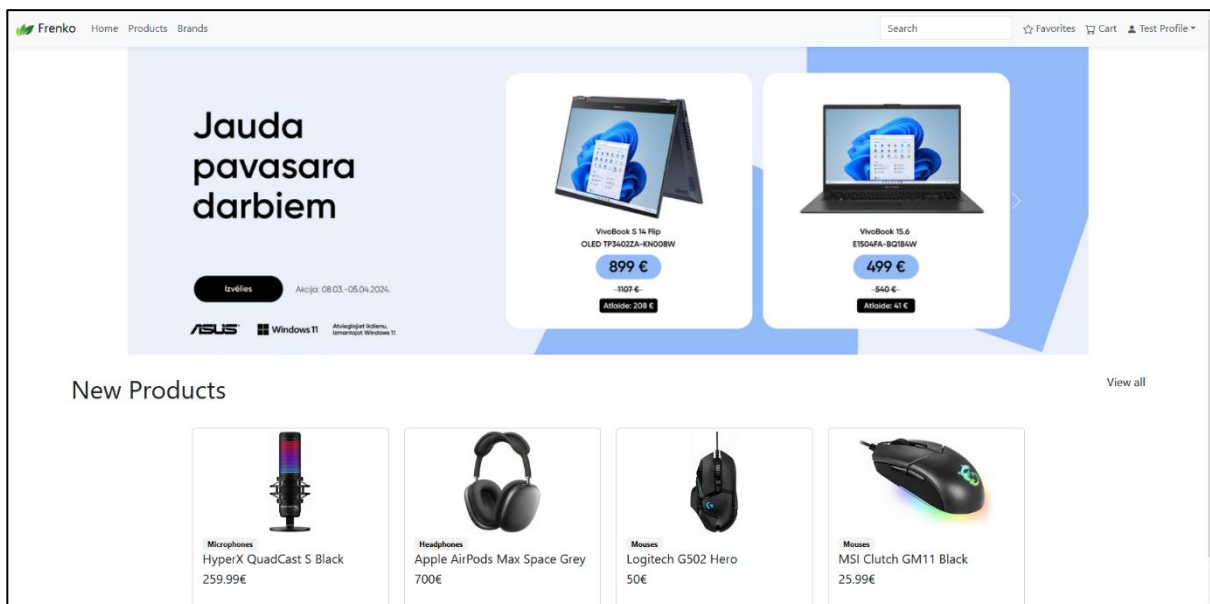
Pēc veiksmīgas migrācijas jāparādās visām nepieciešamajām tabulām zem datubāzes webstore (skat. 6.7. att.).



6.7. att. Attēls ar izveidotām tabulām datubāzē webstore

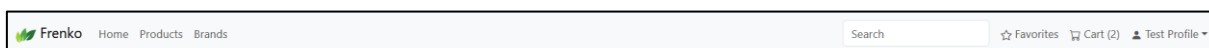
6.3. Programmas apraksts

Sākulapa – Atverot tīmekļa lapu pārlūkprogrammā lietotājam tiks izvadīta sākuma lapa (skat. 6.8. att.). Sākuma lapa atrodas navigācija (skat. 6.9. att.), Reklāmkarogs, kuru lietotājam ir iespēja šķīrstīt, kopā satur trīs reklāmas. Jaunāko produktu sadaļa, kura sastāv no kartiņām (skat. 6.10. att.), poga “View all”, kas novirzīs lietotāju uz sadaļu “Products”.



6.8. att. Sākulapas attēls

Veikala navigācija – atrodas ekrāna pašā augšā (skat. 6.9. att.). neatkarīgi kurā lapā atrodas lietotājs vai kurā lapas pozīcijā tā vienmēr būs redzama vienā un tajā pašā vietā. Ja lietotājs nav pieslēdzies, tad labajā stūrī būs poga “login”, ja lietotājs ir pieslēdzies, tad tajā vietā radīsies lietotāja vārds, uzvārds. Pa kreisi atradīsies pogas “Cart” un “Favorite” kuras novirzīs uz attiecīgajām “Cart” un “Favorite” lapām. Lietotājam nospiežot uz kartiņas “Cart” un vai “Favorite” pie attiecīgajām pogām parādīsies skaitlis ar preču daudzumu grozā vai favorītu sadaļā. Vēl pa kreisi atrodas meklēšana (skat 6.11. att.). Un Kreisajā pusē atrodas pogas “Home”, “Products” (skat 6.14. att.), “Brands” (skat. 6.17. att.) , kuras novirzīs uz attiecīgajām lapām.



6.9. att. Navigācijas joslas attēls

Produkta kartiņa – Produkta kartiņa (skat. 6.10. att.) satur informāciju par Preci – bilde, kategorija, nosaukums, cena. Produkta kartiņā atrodas divas pogas “Cart” un “Favorites”. Uzspiežot Pogu “Cart” attiecīgais produkts tiks pievienots grozam un pogas teksts nomainīsies uz “Remove”, uzpiežot uz pogas ”Remove” prece tiks izņemta no groza. Uzspiežot Pogu “Favorites” attiecīgais produkts tiks pievienots favorītu sadaļai un pogas teksts nomainīsies uz “Remove”, uzpiežot uz pogas ”Remove” prece tiks noņemta no favorītu sadaļas. Uzklikšķinot uz kartiņas atvērsies jauns logs ar plašāku informāciju par produktu (skat. 6.13. att.). Ja prece nebūs pieejama noliktavā, tad pogas “Cart” vietā rādīsies “out of stock”.



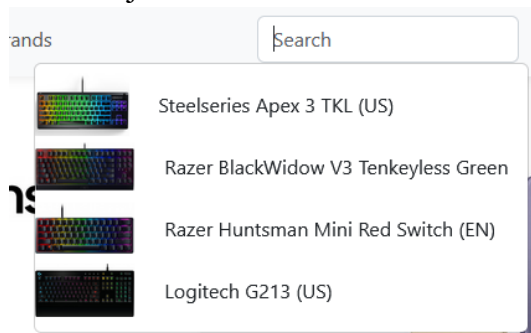
6.10. att. Produkta kartiņa

Globālā meklēšana – Globālā meklēšana (skat. 6.11. att.) atrodas navigācijā labajā pusē.



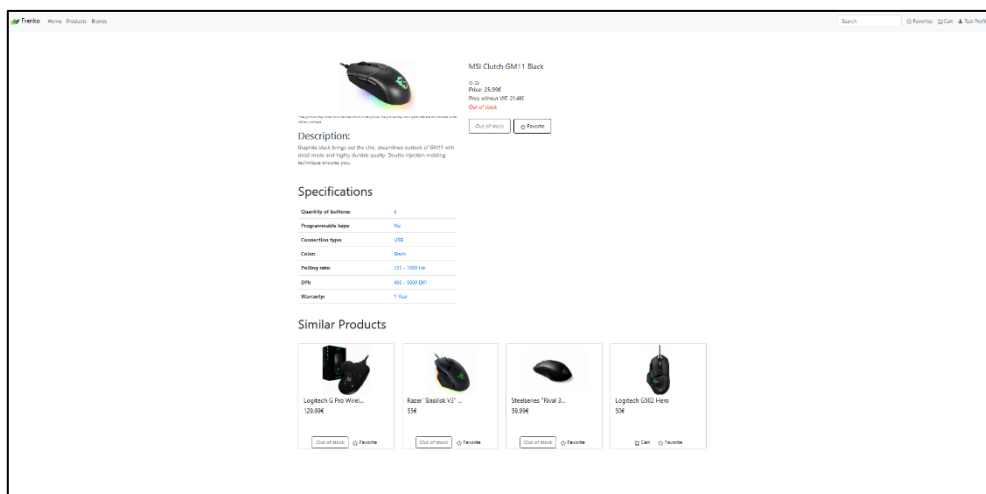
6.11. att. Globālā meklēšana

Uzspiežot uz meklēšanas loga atvērsies “kastīte” (skat. 6.12. att.). zem meklēšanas loga kura piedāvās četrus produktus. Lietotājam meklētājā ievadot preces ID vai nosaukumu tiks piedāvāti produkti, kuri atbilst ievadītajam ID vai nosaukumam.



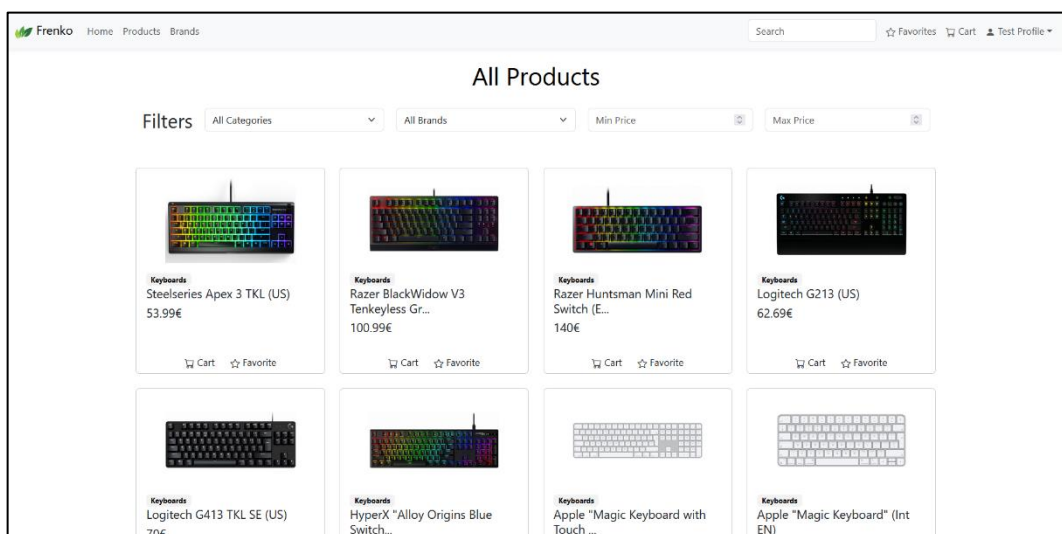
6.12. att. Meklēšanas piedāvājumi

Produkta lapa – Produkta lapa (skat. 6.13. att.). satur informāciju par produktu. Lapas augšējā daļā ir produkta attēls, nosaukums, produkta ID, produkta cena, cena bez PVN, preces daudzums noliktavā, ja prece nebūs noliktavā, tad ar sarkaniem burtiem rādīsies “out of stock”. Zem produkta daudzuma rādīsies divas pogas “Favorite” un “Cart”, spiežot “Car” prece tiks pievienota grozam, ja prece nebūs pieejama rādīsies “out of stock”, spiežot “Favorite”, prece tiks pievienota favorītiem. Zem bildes rādīsies paziņojums, ka bilde ir tikai ilustratīva. Zem paziņojuma rādīsies produkta apraksts un tā specifikācija. Zem preces specifikācijas atradīsies četras līdzīgu produktu kartiņas.



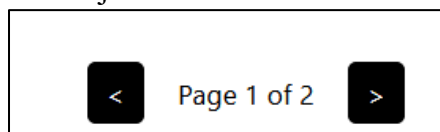
6.13. att. Produkta lapa

Visu produktu lapa – Visu produktu lapā (skat. 6.14. att.) atradīsies kartiņas ar visiem pieejamajiem produktiem veikalā. Lapā vienlaicīgi tiks atrādīti divpadsmit produkti, pēc tam tiks rādīts lapaspušu pārslēgs (skat. 6.15. att.) ar kuru var pāriet uz nākamo lapu. Lapā arī atrodas filtrēšanas opcijas (skat. 6.16. att.) ar kuru palīdzību ir iespējams filtrēt kartiņas.



6.14. att. Visu produktu lapa

Lapaspušu pārslēgs – Lapaspušu pārslēgs (skat. 6.15. att.) parādās lapās, kur pēc kārtas ir vismaz trīspadsmit kartiņa, jo vienā lapā optimālai skatīšanai ir paredzētas tikai divpadsmit kartiņas. Uzspiežot bultiņu pa labi lietotājs tiks novirzīts uz nākamo lapu ar pārējām kartiņām.



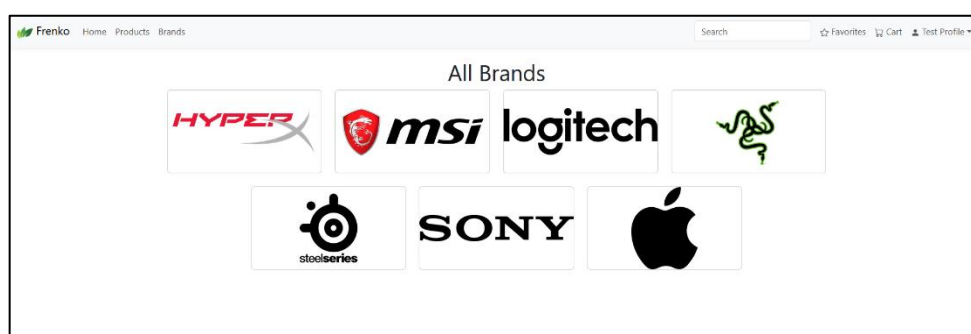
6.15. att. Lapaspušu pārslēgs

Filtri – Filtri (skat. 6.16. att.) satur iespēju sevī filtrēt produktu kartiņas, katrs filtrs papildina iepriekšējo tādējādi dodot stiprāku filtrēšanas opciju. Filtriem ir opcijas filtrēt pēc – kategorijas, zīmola, minimālās cenas un maksimālās cenas.



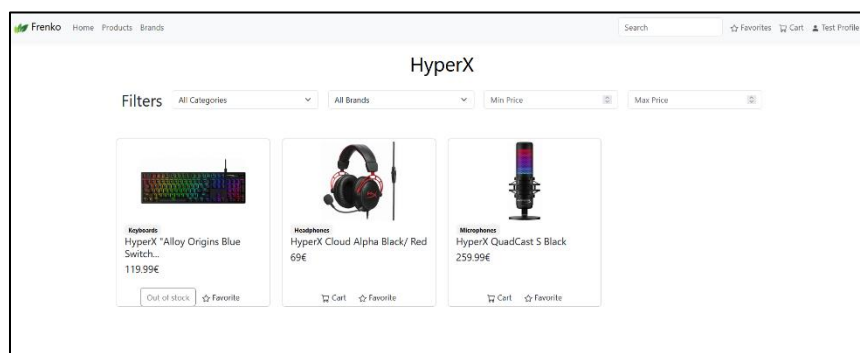
6.16. att. Filtrēšanas opcijas

Visi ražotāji – Lapa visi ražotāji (skat. 6.17. att.) satur datus par visiem ražotājiem, kuru produkti atrodas veikalā uzspiežot uz ražotāja atvērsies produkta lapa kur pieejami tikai konkrētā ražotāja produkti.



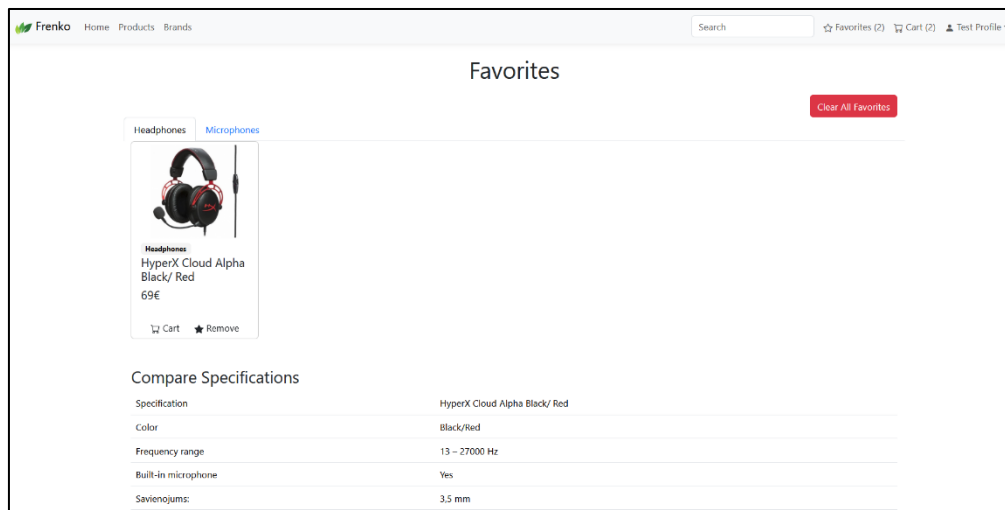
6.17. att. Visi ražotāji

Ražotāja produkti – Lapa ražotāja produkti (skat. 6.18. att.) saturu produktu labu, tikai vienīgi ir pieejamas konkrētā ražotāja preces.



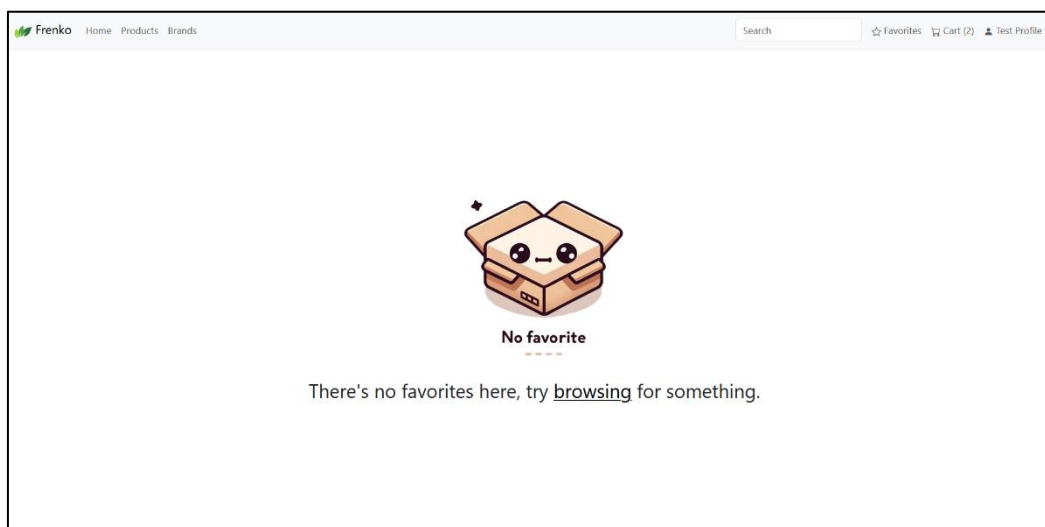
6.18.att. Ražotāja produkti

Favorītu lapa – favorītu lapa (skat. 6.19. att.) satur informāciju par produktiem kurus lietotājs ir atzīmējis ar norādi “favorite”. Lapā preces tiek kategorizētas pa kategorijām. Katras kategorijas preces tiek savā starpā salīdzinātas. Lietotājs preci var noņemt no šīs lapas uz preces kartiņas spiežot “Remove” vai arī spiežot lielo sarkano pogu “Clear all favorites”, tādējādi noņemot visas kartiņas no šī loga.



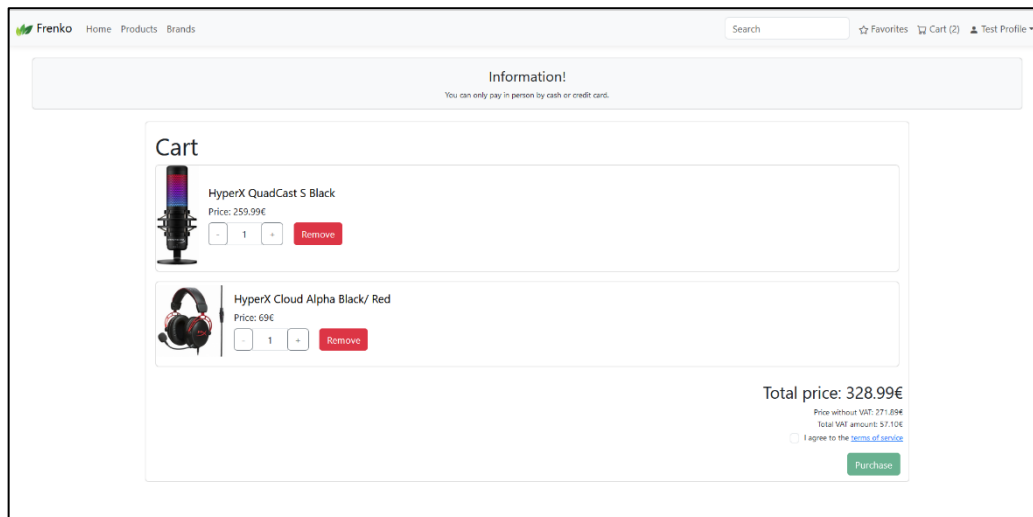
6.19.att. Favorītu lapa

Favorītu lapa (preces) – Ja neviena prece nav pievienota favorītu lapai (skat. 6.20. att.), tad rādīsies, ka neviena prece nav pievienota, tekstā būs iekrāsots vārds, uz kura nospiežot lietotājs tiks novirzīts uz produktu lapu.



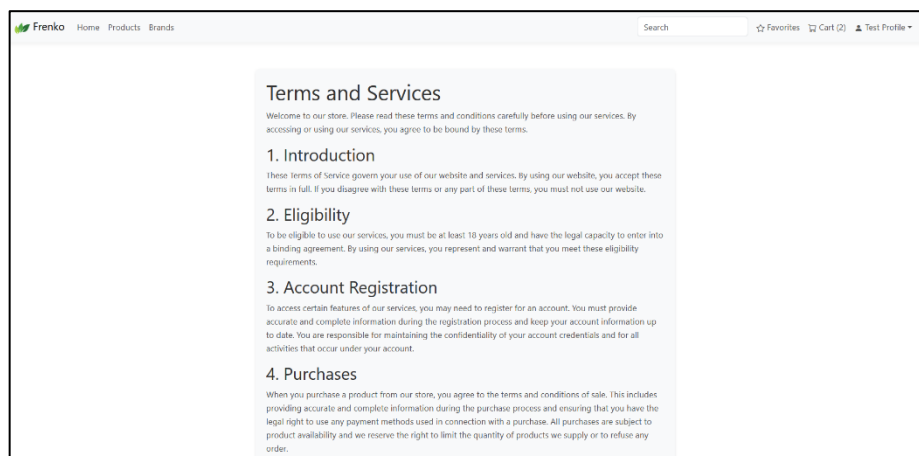
6.20.att. Favorītu lapa

Grozs – Lapā grozs (skat. 6.21. att.) tiks atrādītas preces, kuras lietotājs ir ielicis grozā spiežot uz pogas “Cart”. Lapā augšā ir izvadīta informācija par to, ka maksāt par precēm var tikai klātienē. Zemāk ir pirkuma groza preces. Lietotājam ir iespēja noņemt preci, ja to nevēlas, vai uzspiežot + vai – pievienot vairāk vai mazāk preces. Virs pogas “Purchase” atrodas informācija par gala summu, cenu bez PVN un PVN summu. Lai nopirktu preces, kuras atrodas grozā lietotājam ir jāatķeksē, ka ir izlasījis pakalpojuma sniegšanas noteikumus (skat. 6.22. att.), tikai tad atbloķēsies poga “Purchase” un lietotājam būs iespēja nopirkt preces.



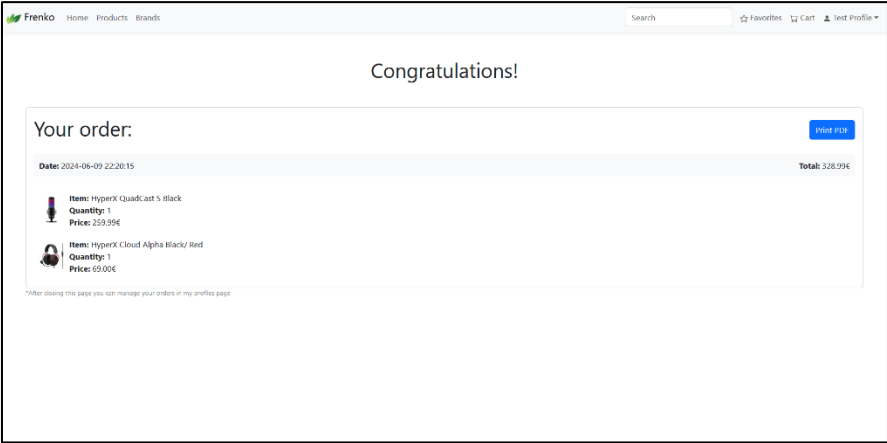
6.21.att. Groza lapas skats

Pakalpojuma sniegšanas noteikumi – Lapā pakalpojuma sniegšanas noteikumi (skat. 6.22. att.) tiks atrādīti pakalpojuma sniegšanas noteikumi.




6.22.att. Pakalpojuma lietošanas noteikumi

Paldies lapa – Pēc pirkuma veikšanas parādīsies paldies lapa (skat. 6.23. att.). Lapā tiks izvadīta informācija par iegādātajām precēm, to daudzums cena, nosaukums un attēls. Loga kreisajā augšējā stūrī atrodas pirkuma laiks, bet labajā poga “Print PDF”, kura izprintēs pirkuma kvīti (skat. 6.24. att.).



6.23.att. Paldies lapa

Pirkuma kvīts – Pirkuma kvītī (skat. 6.24. att.) būs apskatāms PDF formātā, ko lietotājs ir iegādājies. Kvītī būs dati par lietotāju, pirkumu, pirkuma laiku. Un infromācija cik ilgā laikā jāsamaksā.

 **Frenko**

Purchased At: Invalid Date Invalid Date

Buyer: Test Profile
Email: test@gmail.com
Phone: 25966512

Item Name	Quantity	Price without VAT (€)	VAT Amount (€)	VAT %	Total Price (€)
HyperX QuadCast S Black	1	214.87	45.12	21.00	259.99
HyperX Cloud Alpha Black/ Red	1	57.02	11.98	21.00	69

Description	Amount (€)
Total Price without VAT	271.89
Total VAT Amount	57.10
Total Price	328.99

Payment Terms and Conditions
Payment is due within 30 days from the purchase date. After 30 days purchases will be canceled!

Issuer: _____ Receiver: _____

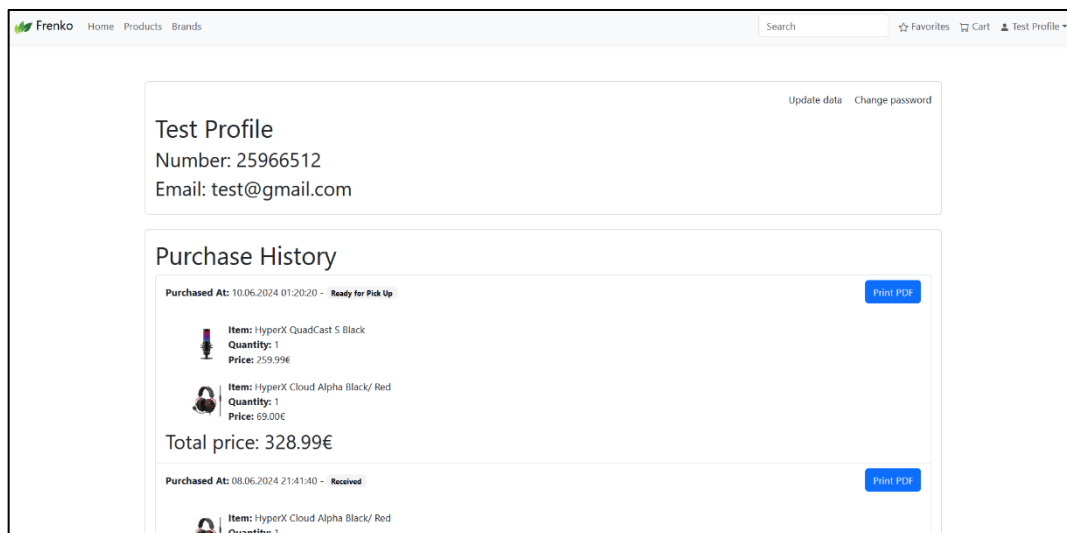
Name Surname: _____ Name Surname: _____

Date: _____ Date: _____

Signature: _____ Signature: _____

6.24.att. Pirkuma kvīts

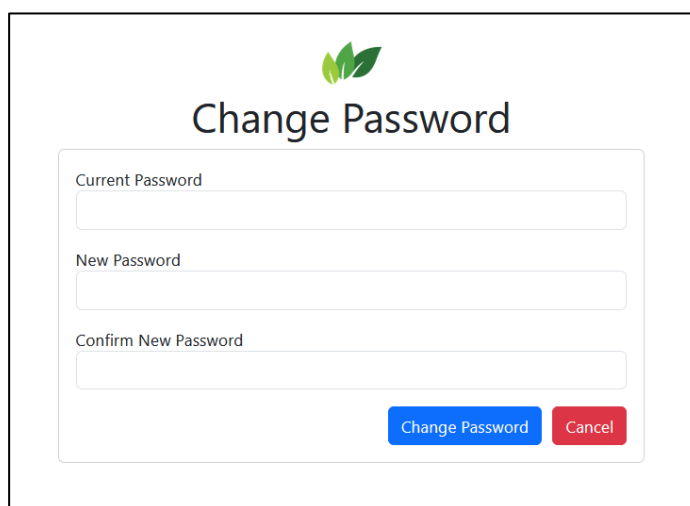
Man profils – Sadaļā mans profils (skat. 6.25. att.) ir apskatāma informācija par lietotāju un veiktajiem pirkumiem. Sadaļā lietotājam ir iespēja mainīt paroli (skat. 6.26. att.) un atjaunot profila informāciju (skat. 6.27. att.). Zem Profila datiem lietotājs var apskatīt savu pirkumu vēsturi (skat. 6.28. att.).



The screenshot shows the 'Test Profile' page on the Frenko website. The page has a header with the Frenko logo and navigation links (Home, Products, Brands). The main content area is divided into two sections. The top section, titled 'Test Profile', displays the user's name 'Test Profile', a number '25966512', and an email 'test@gmail.com'. There are links for 'Update data' and 'Change password'. The bottom section, titled 'Purchase History', shows a list of purchases. The first purchase is dated '10.06.2024 01:20:20' and is marked 'Ready for Pick Up'. It lists two items: 'HyperX QuadCast S Black' (Quantity: 1, Price: 259.99€) and 'HyperX Cloud Alpha Black/ Red' (Quantity: 1, Price: 69.00€). The total price is 328.99€. There is a 'Print PDF' button next to the purchase history. The second purchase is dated '08.06.2024 21:41:40' and is marked 'Received'. It lists one item: 'HyperX Cloud Alpha Black/ Red' (Quantity: 1).

6.25.att. Profila lapa

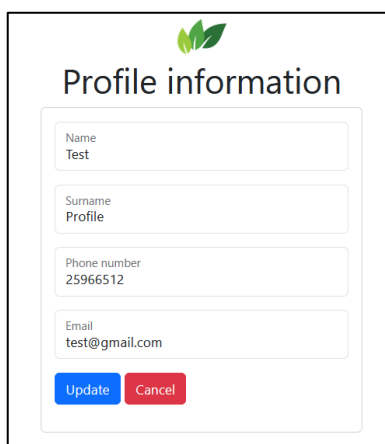
Mainīt paroli – Lapā mainīt paroli (skat. 6.26. att.) lietotājam ir iespēja nomainīt savu esošo paroli pret citu, uzspiežot pogu “cancel” lietotājs tiks novirzīts atpakaļ un lapu mans profils, uzspiežot pogu “change password” parole tiks nomainīta vai parādīts kļūdas paziņojums



The screenshot shows the 'Change Password' form on the Frenko website. The form has a title 'Change Password' with a green leaf icon above it. It contains three input fields: 'Current Password', 'New Password', and 'Confirm New Password'. At the bottom right of the form are two buttons: 'Change Password' (blue) and 'Cancel' (red).

6.26.att. Paroles mainīšana lapa


Profila datu atjaunošana – Profila datu atjaunošanas lapā (skat. 6.27. att.) lietotājs var atjaunot datus par sevi, ja tie ir mainījušies.



The screenshot shows a 'Profile information' form with a green leaf icon at the top. The form contains four input fields: 'Name' with the value 'Test', 'Surname' with the value 'Profile', 'Phone number' with the value '25966512', and 'Email' with the value 'test@gmail.com'. At the bottom of the form are two buttons: a blue 'Update' button and a red 'Cancel' button.

6.27.att. Profila datu atjaunošanas lapā

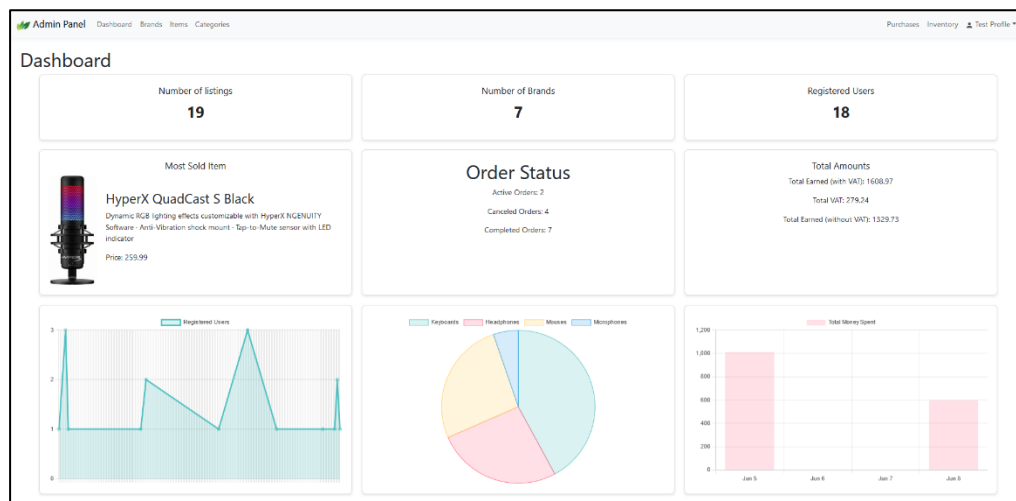
Pirkuma vēsture – Sadaļā pirkuma vēsture (skat. 6.28. att.) var apskatīt iepriekšējos pirkumus, kādas preces pirktas, pa kādu cenu un kādā daudzumā, var apskatīt kad pasūtījums veikts un kāds ir tā statuss, kā arī ir iespējams izprintēt kvīti.



The screenshot shows the 'Purchase History' page. At the top, it says 'Purchased At: 10.06.2024 01:20:20 - Ready for Pick Up' and has a 'Print PDF' button. Below this, there are two items listed: 'HyperX QuadCast S Black' with a quantity of 1 and a price of 259.99€, and 'HyperX Cloud Alpha Black/ Red' with a quantity of 1 and a price of 69.00€. At the bottom, the 'Total price' is 328.99€.

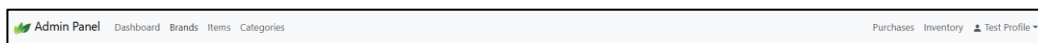
6.28.att. Pirkuma vēsture

Vadības panelis – Vadības panelis (skat. 6.29. att.) satur informāciju par veikalu – statistiku. Panelī atrodas diagrammas par datiem no veikala, reģistrēto lietotāju skaits, dienas ieņēmumi, kategorijas un cik daudz preces ir katrā kategorijā, preču kopskaitu un zīmolu kopskaitu, pārdotāko preci, Pasūtījuma statusiem un kopējo peļņu.



6.29.att. Vadības panelis

Administrators rīku navigācija – Administratoru rīku navigācija (skat. 6.30. att.) pēc izskata ir tāda pati, kā parastā, tikai tai ir papildus pogas un pazūd meklēšanas lodziņš.



6.30.att. Administrators rīku navigācija

Ražotāji – Lapā ražotāji (skat. 6.31. att.) atrodas tabula ar visiem ražotājiem, tajā atrodas poga “Add Brand” ražotājus, kura ved uz lapu pievienot ražotājus (skat. 6.32. att.), Poga eksportē uz Excel, kura eksportē (skat. 6.33. att.) visus atzīmētos laukus uz Excel tabulu un meklēšanas logs, kur iespējams meklēt ierakstu pēc nosaukuma vai ID.

Brand ID	Brand Name	Brand Image Path	Brand Image	Item Count	Actions
45	HyperX	JKkOnF7DEV.png		3	<input type="checkbox"/> Edit Delete
46	MSI	IMziQDo9Ed.png		1	<input type="checkbox"/> Edit Delete
47	Logitech	7OY7b15ipr.png		5	<input type="checkbox"/> Edit Delete
61	Razer	oJolhcv6ee.png		3	<input type="checkbox"/> Edit Delete
62	SteelSeries	z9ncaasonp.png		3	<input type="checkbox"/> Edit Delete
67	Sony	DfWXea3LcU.png		1	<input type="checkbox"/> Edit Delete
71	Apple	UwlGavev6s.png		3	<input type="checkbox"/> Edit Delete

6.31.att. Ražotāju lapa

Pievienot Ražotāju – Lapā pievienot ražotāju (skat. 6.32. att.) ir iespējams pievienot ražotāju ievadot tā nosaukumu un attēlu. Ja lietotājs uzspiedīs uz pogas “Save” tad tiks novirzīts atpakaļ uz Ražotāju lapu un Ražotājs tiks saglabāts, ja uzspiedīs “cancel”, tad tiks novirzīts atpakaļ, bet ieraksts netiks saglabāts.

6.32.att. Pievienot ražotāju lapa

Eksports uz Excel – Izvēloties konkrētus ierakstus vai visus un nospiežot pogu “export” dati tiks eksportēti uz excel (skat. 6.33. att.).

	A	B	C	D
1	id	name	imgPath	
2	45	HyperX	JKkOnF7DEV.png	
3	46	MSI	IMziQDo9Ed.png	
4	47	Logitech	7OY7b15ipr.png	
5	61	Razer	oJolhcv6ee.png	
6	62	SteelSeries	z9ncaasonp.png	
7	67	Sony	DfWXea3LcU.png	
8	71	Apple	UwlGavev6s.png	

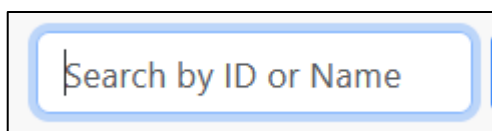
6.33. att. Eksports uz Excel

Ziņojums par veiksmīgu Pievienošanu, Rediģēšanu vai Dzēšanu – Veicot veiksmīgu Pievienošanu, Rediģēšanu vai Dzēšanu parādīsies paziņojums (skat. 6.34. att.). par to.



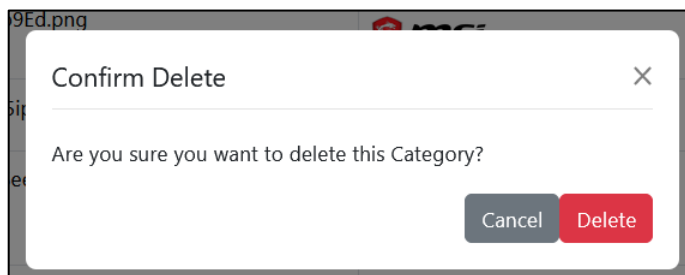
6.34. att. Ziņojums par veiksmīgu darbību

Ierakstu meklēšana tabulās – Ierakstus tabulās var meklēt ar meklētāja (skat. 6.35. att.) palīdzību ievadot ID vai nosaukumu.



6.35.att. Ierakstu meklētājs

Ieraksta dzēšana – Administratoram ir iespēja dzēst ierakstus, bet pirms tam parādīsies kartīte, vai tiešām vēlaties to darīt (skat. 6.36. att.). Uzspiežot pogu “cancel” nekas nenotiks un kartīte tiks aizvērota. Uzspiežot pogu “delete” ieraksts tiks izdzēsts.









6.36.att. Vai tiešām vēlaties dzēst kartīte

Rediģēt ražotāju – Administratoram ir iespēja rediģēt ražotāju (skat. 6.37. att.). Uzspiežot pogu “cancel” administrators tiks atgriezts ražotāju lapā. Ievadot kaut ko kādā no laukiem un uzspiežot pogu “save” ieraksts tiks rediģēts.

A form titled "Edit Brand". It has two input fields: "Name" with the value "Apple" and "Image" with a "Browse..." button and the text "No file selected.". At the bottom, there are two buttons: "Update" (blue) and "Cancel" (red).

6.37.att. Vai tiešām vēlaties dzēst kartīte

Preces – Preces (skat. 6.38. att.) satur informāciju par visām pievienotajām precēm – to nosaukums, apraksts, cena, bildes ceļš datubāzē, pati bilde. Panelī atrodas poga “Add item” – vedīs uz sadaļu kur iespējams pievienot jaunu preci, poga “Edit” vedīs uz sadaļu labot preces informāciju, poga “delete” dzēsīs preci. Sadaļā arī atrodas poga eksportēt uz Excel, kura pēc noklusējuma ir izslēgta, tā ieslēgsies, kad tik atlasītas visas preces, vai kāda konkrēta prece vai preces. Uzspiežot uz pogas eksportēt tiks izveidots jauns Excel fails ar tabulas saturu. Sadaļā atrodas Pārejas, kur iespējams staigāt pa tabulām.

Items							
Search by ID or Name							
Add Item Export Selected Rows							
Items	Specification Descriptions						
<input type="checkbox"/>	Item ID	Name	Description	Price	Img path	Img	Actions
<input type="checkbox"/>	38	SteelSeries Apex 3 TKL (US)	World's First Water-Resistant TKL Keyboard: SteelSeries Apex 3 TKL	53.99€	CKShjD5bNv.jpg		Edit Delete
<input type="checkbox"/>	39	Razer BlackWidow V3 Tenkeyless Green	Compact gaming keyboard featuring Razer Mechanical Switches, customizable lighting powered by Razer Chroma RGB, and aluminum construction.	100.99€	DXLrSaOKkb.jpg		Edit Delete
<input type="checkbox"/>	40	Razer Huntsman Mini Red Switch (FN)	Featuring Razer's cutting-edge Optical Switches that offers a lightning-fast actuation in a smaller form factor designed for maximum portability.	140€	GjmkALBdm.jpg		Edit Delete
<input type="checkbox"/>	41	Logitech G213 (US)	The G213 gaming keyboard features Logitech G Mech Dome keys that are specially tuned to deliver a superior tactile response and overall performance profile.	62.69€	453Hxull1.jpg		Edit Delete
<input type="checkbox"/>	42	Logitech G413 TKL SE (US)	G413 TKL SE puts gaming-grade performance—from tactile mechanical switches to 6-key rollover anti-ghosting and PBT keycaps—into optimal compact form.	70€	ppAYXDK3gA.jpg		Edit Delete
<input type="checkbox"/>	43	HyperX Alloy Origins Blue Switches (FN)	The HyperX Alloy Origins is a mechanical gaming keyboard with a compact design and customizable RGB key lighting. Take it with	119.99€	HjYroi0EQ.jpg		Edit Delete

6.38.att. Preču lapa

Pievienot preci – Pievienojot preci atvērsies jauns logs (skat. 6.39. att.), kur lietotājs varēs ievadīt informāciju par preci. Izvēlēties kategoriju, ražotāju, ievadīt preces nosaukumu, aprakstu, cenu, attēlu. Uzspiežot pogu “cancel” administrators tiks novirzīts atpakaļ uz preču lapu un ieraksts nebūs saglabāts

Add Item

Category

Brand

Name

Description

Price

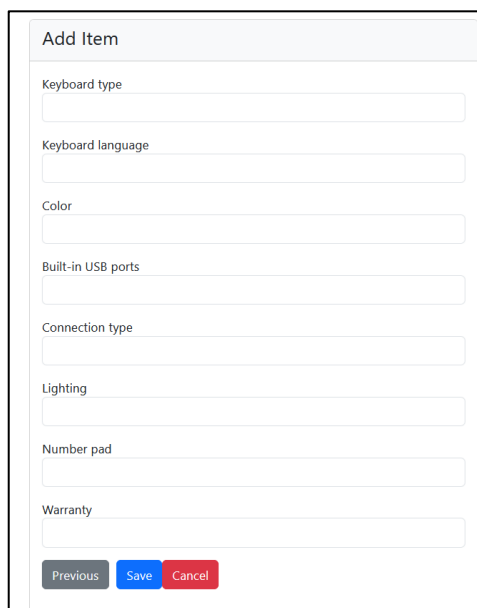
Image

Browse... No file selected.

Next Cancel

6.39.att. Pievienot preces

Uzspiežot pogu “next” atvērsies nākamais logs (skat. 6.40. att.), kur jāievada preces specifikācija. Ja administrators uzspiedīs pogu “cancel” administrators tiks novirzīts atpakaļ uz preču lapu un ieraksts nebūs saglabāts



6.40.att. Pievienot preces specifikāciju

Rediģēt preces – Uzspiežot pogu “Edit” atvērsies logs (skat. 6.41. att.), kur lietotājam ir iespēja rediģēt preces ievadot aizpildītajos laukos jaunas vērtības. Uzspiežot pogu “cancel” administrators tiks atgriezts preču lapā. Ievadot kaut ko kādā no laukiem un uzspiežot pogu “save” ieraksts tiks rediģēts.



6.41.att. Rediģēt preci lapa

Specifikāciju apraksti – Sadaļā specifikāciju apraksti (skat. 6.42. att.) ir iespējams izvēlēties konkrētu kategoriju, izvēloties kategoriju tiks atvērta tabula ar visiem specifikāciju aprakstiem konkrētai prece konkrēta kategorijā.

Items

Search by ID or Name

Add Item

Export Selected Rows

Items

Specification Descriptions

Keyboards

Mouses

Headphones

Microphones

<input type="checkbox"/>	Item ID	Item Name	Keyboard type	Keyboard language	Color	Built-in USB ports	Connection type	Lighting	Number pad	Warranty
<input type="checkbox"/>	38	Steelseries Apex 3 TKL (US)	Gaming	EN (US)	Black	No	USB	No	No	2 Years
<input type="checkbox"/>	39	Razer BlackWidow V3 Tenkeyless Green	Gaming	EN, RU	Black	No	USB	Yes	No	2 Years
<input type="checkbox"/>	40	Razer Huntsman Mini Red Switch (EN)	Gaming	EN (US)	Black	No	USB	Yes	No	2 Years
<input type="checkbox"/>	41	Logitech G213 (US)	Gaming	ES (US)	Black	No	USB	Yes	Yes	2 Years
<input type="checkbox"/>	42	Logitech G413 TKL SE (US)	Gaming	EN (US)	Black	No	USB	No	No	2 Years
<input type="checkbox"/>	43	Hyperx "Alloy Origins Blue Switches" (EN)	Gaming	EN (US)	Black	No	USB	Yes	Yes	2 Years
<input type="checkbox"/>	44	Apple "Magic Keyboard with Touch ID and Numeric Keypad" (RU)	Classic	EN, RU	White	No	Bluetooth	No	Yes	1 Year
<input type="checkbox"/>	45	Apple "Magic Keyboard" (Int EN)	Classic	EN	White	No	Bluetooth	No	No	1 Year

6.42.att. Preču specifikāciju apraksti

Kategorijas – Lapā kategorijas (skat. 6.43. att.) ir iespējams apskatīt, pievienot un dzēst kategorijas, kā arī tās meklēt un eksportēt.

Categories				
Search by ID or Name				
Add Category Export Selected Rows				
Categories Specification Titles				
<input type="checkbox"/>	Category ID	Category Name	Item Count	Actions
<input type="checkbox"/>	29	Keyboards	8	Edit Delete
<input type="checkbox"/>	30	Headphones	5	Edit Delete
<input type="checkbox"/>	31	Mouses	5	Edit Delete
<input type="checkbox"/>	32	Microphones	1	Edit Delete

6.43.att. Kategoriju lapa

Kategoriju specifikācijas virsraksti – Kategoriju specifikācijas virsrakstu lapā (skat. 6.44. att.) iespējams apskatīt rediģēt un dzēst kategoriju specifikāciju virsrakstus, virsrakstus iespējams apskatīt katrai kategorijai atsevišķi pārvietojoties pa tabulām.

Categories			
Search by ID or Name			
Add Specification Title Export Selected Rows			
Categories Specification Titles			
Keyboards Headphones Mouses Microphones			
<input type="checkbox"/>	Specification Title ID	Specification Title	Actions
<input type="checkbox"/>	23	Keyboard type	Edit Delete
<input type="checkbox"/>	24	Keyboard language	Edit Delete
<input type="checkbox"/>	25	Color	Edit Delete
<input type="checkbox"/>	26	Built-in USB ports	Edit Delete
<input type="checkbox"/>	27	Connection type	Edit Delete
<input type="checkbox"/>	28	Lighting	Edit Delete
<input type="checkbox"/>	29	Number pad	Edit Delete
<input type="checkbox"/>	30	Warranty	Edit Delete

6.44.att. Kategoriju specifikāciju virsrakstu lapa

Kategoriju virsrakstu rediģēšana – Kategoriju virsrakstu rediģēšanas lapā (skat. 6.45. att.) iespējam rediģēt virsrakstu ievadot izmaiņas ievades logā. Uzspiežot pogu “cancel” administrators tiks atgriezts kategoriju virsrakstu lapā. Ievadot kaut ko kādā no laukiem un uzspiežot pogu “save” ieraksts tiks rediģēts.



6.45.att. Kategoriju virsrakstu rediģēšanas lapā

Kategorijas rediģēšana – Kategoriju rediģēšanas lapā (skat. 6.46. att.) iespējam rediģēt kategoriju ievadot izmaiņas ievades logā. Uzspiežot pogu “cancel” administrators tiks atgriezts kategoriju virsrakstu lapā. Ievadot kaut ko kādā no laukiem un uzspiežot pogu “save” ieraksts tiks rediģēts.



6.46. att. Kategoriju rediģēšanas lapā

Pievienot kategoriju - Pievienot kategoriju lapā (skat. 6.47. att.) iespējams pievienot kategoriju un tās specifikācijas virsrakstus. Lietotājs var spiest pogu “Add more”, lai pievienotu jaunu lauku priekš specifikācijas virsraksta ievades. Spiežot pogu “Remove” ievades lauks tiks noņemts, bet vienmēr būs jāievada vismaz viens specifikācijas virsraksts, lai būtu iespējams saglabāt kategoriju. Uzspiežot pogu “cancel” administrators tiks atgriezts kategoriju virsrakstu lapā. Ievadot kaut ko kādā no laukiem un uzspiežot pogu “save” ieraksts tiks rediģēts.



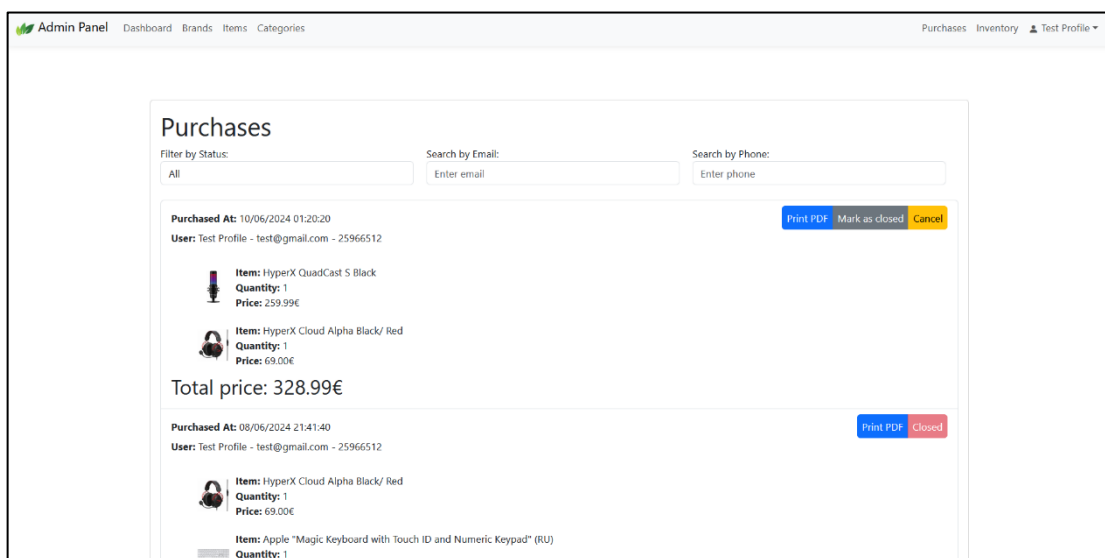
6.47.att. Kategoriju pievienošana

Kļūdas paziņojums – Kļūdas paziņojumi (skat. 6.48. att.) tiks izvadīti, kad lietotājs nepareizi būs ievadījis datu, vai nebūs tos ievadījis vispār. Kļūdas paziņojumus būs iespējams sastapt gandrīz visās mājaslapas sadaļās.

- The name field is required.
- The description field is required.
- The price field is required.
- The selected brand id is invalid.
- The selected categories id is invalid.
- The img must be an image.

6.48. att. Kļūdas paziņojums

Lietotāju pirkumi – Lietotāju pirkumi lapā (skat. 6.49. att.) tiks izvadīta informācija par lietotāju veiktajiem pirkumiem. Lapā atrodas meklēšanas un filtrēšanas opcijas (skat. 6.50. att.), kā arī kartītes ar veiktajiem pirkumiem opcijas (skat. 6.51. att.).



6.49.att. Lietotāju pirkumu lapā

Veikto pirkumu meklēšana, filtrēšana – Veikto pirkumu meklēšanā, filtrēšanā (skat. 6.50. att.) ir iespējams meklēt pirkumus pēc to statusiem (“active”, “closed”, “canceled”). Ir iespējams meklēt pasūtījumus arī pēc pasūtītāja telefona numura vai e-pasta adreses.



6.50.att. Veikto pirkumu meklēšana, filtrēšana

Lietotāju pirkumu kartītes – Lietotāju pirkumu kartītēs (skat. 6.51. att.) administratoram būs iespēja apskatīties, ko lietotājs ir iegādājies, apskatīt pirkuma statusu un mainīt pirkuma statusu atkarībā no nepieciešamības. Pirkuma statusu ir iespējams mainīt uz “closed” un “canceled”. “Closed”, ir kad pirkums ir apmaksāts un saņemts, “canceled” ir kad pirkums ir atcelts. Administratoram arī ir iespēja Izprintēt pirkuma kvīti. Kā arī administrators redz lietotāja vārdu, uzvārdu, e-pastu, telefonu numuru, ar kuru veikts pasūtījums

Purchased At: 10/06/2024 01:20:20
User: Test Profile - test@gmail.com - 25966512

[Print PDF](#)
[Mark as closed](#)
[Cancel](#)


Item: HyperX QuadCast S Black
Quantity: 1
Price: 259.99€


Item: HyperX Cloud Alpha Black/ Red
Quantity: 1
Price: 69.00€

Total price: 328.99€

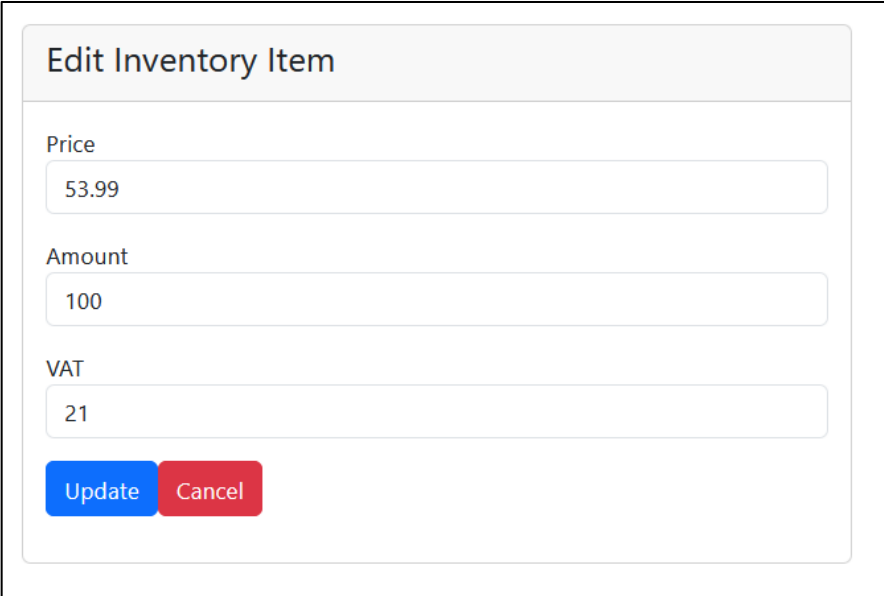
6.51.att. Lietotāju pirkumu kartītes

Preču pārvaldības lapa – Preču pārvaldības lapā (skat. 6.52. att.) administratoram būs iespēja apskatīt cik preces ir rezervētas, nopirkas un cik preces atrodas noliktavā, tā pat būs iespēja meklēt preces pēc ID un nosaukuma, kā arī būs iespēja eksportēt uz Excel. Administratoram arī būs iespēja rediģēt preces cenu, daudzumu un nodokli uzspiežot uz pogas “edit”.

Admin Panel Dashboard Brands Items Categories Purchases Inventory Test Profile								
Inventory								
Search by ID or Name Export Selected Rows								
<input type="checkbox"/>	ID	Name	Price	Vat	Amount	Reserved	Sold	Actions
<input type="checkbox"/>	38	Steelseries Apex 3 TKL (US)	53.99€	21%	100	0	0	Edit
<input type="checkbox"/>	39	Razer BlackWidow V3 Tenkeyless Green	100.99€	21%	100	0	0	Edit
<input type="checkbox"/>	40	Razer Huntsman Mini Red Switch (EN)	140€	21%	100	0	0	Edit
<input type="checkbox"/>	41	Logitech G213 (US)	62.69€	21%	100	0	0	Edit
<input type="checkbox"/>	42	Logitech G413 TKL SE (US)	70€	21%	50	0	1	Edit
<input type="checkbox"/>	43	HyperX "Alloy Origins Blue Switches" (EN)	119.99€	21%	0	0	0	Edit
<input type="checkbox"/>	44	Apple "Magic Keyboard with Touch ID and Numeric Keypad" (RU)	199.99€	21%	25	0	1	Edit
<input type="checkbox"/>	45	Apple "Magic Keyboard" (Int EN)	100€	21%	30	0	0	Edit
<input type="checkbox"/>	46	Sony WH-1000XM4 Black	259€	21%	25	0	0	Edit
<input type="checkbox"/>	47	HyperX Cloud Alpha Black/ Red	69€	21%	3	1	1	Edit
<input type="checkbox"/>	48	Steelseries Arctis Nova 1 Black	59.99€	21%	0	0	0	Edit
<input type="checkbox"/>	49	Logitech G435 Lightspeed Wireless Headset White	70.99€	21%	13	0	0	Edit

6.52.att. Preču pārvaldības lapa

Preču pārvaldīšanas rediģēšanas lapa – Preču pārvaldīšanas rediģēšanas lapā (skat. 6.53. att.) var veikt izmaiņas par preci. Ir iespējams norādīt Preces cenas izmaiņas, preču daudzuma izmaiņas un pievienotā vērtības nodokļa izmaiņas.



Edit Inventory Item

Price

53.99

Amount

100

VAT

21

Update Cancel

6.53.att. Preču pārvaldīšanas rediģēšanas lapa

Finanšu pārskats – Finanšu pārskats lapā (skat. 6.54. att.) administratoram būs iespēja apskatīt finanšu pārskatu pa kategorijām un pa precēm. Lapā būs apskatāms, cik katra kategorija un cik katra preces ir ietirgojusi.

Financial Overview

By Category

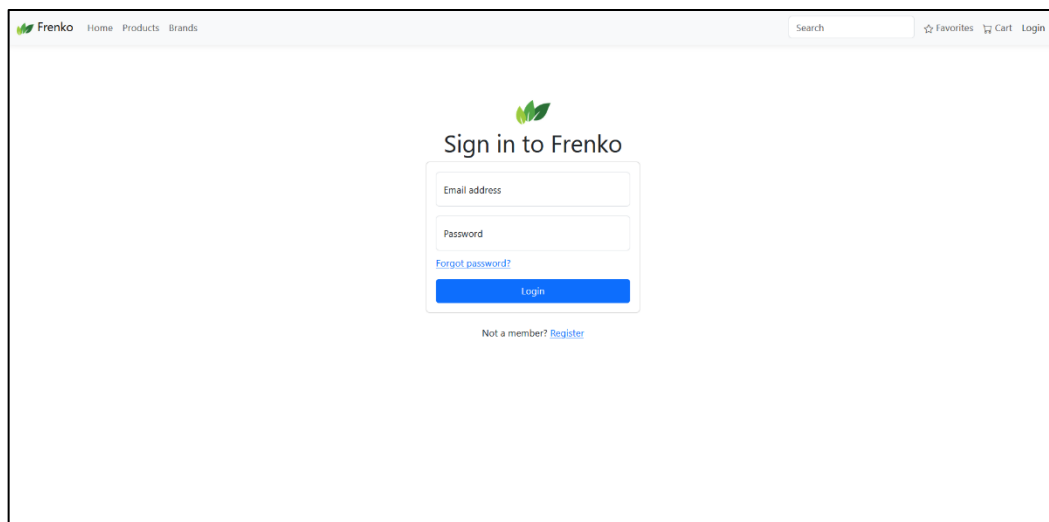
By Item

By Category

Category ID	Category	Total Earned (with VAT)	Total VAT	Total Earned (without VAT)
31	Mouses	50.00	8.68	41.32
30	Headphones	769.00	133.46	635.54
32	Microphones	519.98	90.24	429.74
29	Keyboards	269.99	46.86	223.13
Total		1608.97	279.24	1329.73

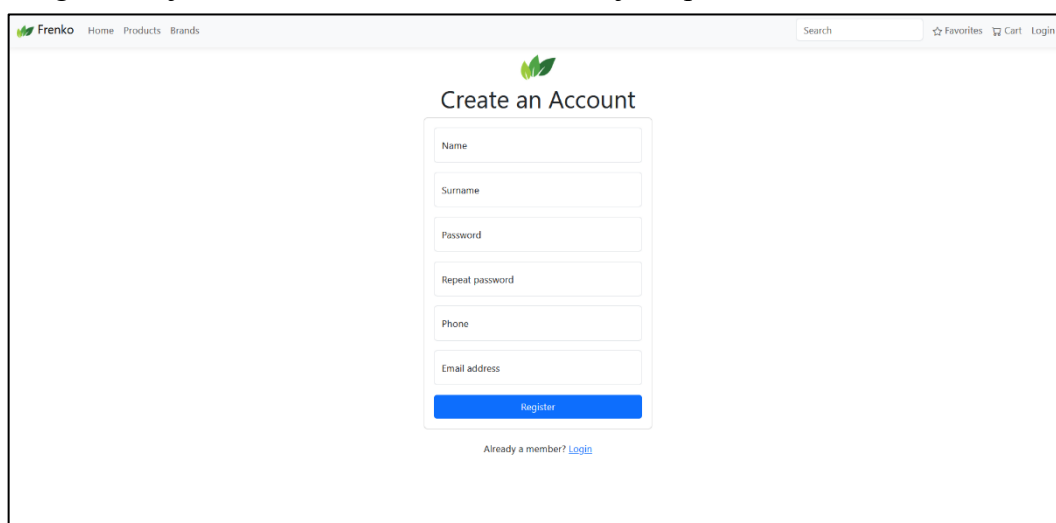
6.54.att. Finanšu pārskata lapa

Autorizēšanās lapa – Autorizēšanās lapā (skat. 6.55. att.) lietotājiem ir iespēja autorizēties ievadot pareizu e-pastu un paroli, ja dati būs pareizi lietotājs būs sekmīgi autorizējies, ja dati nebūs pareizi tiks izvadīts kļūdas paziņojums. Lietotājiem no šīs lapas arī ir iespējams doties uz reģistrācijas lapu spiežot uz iekrāsotā teksta “Register”



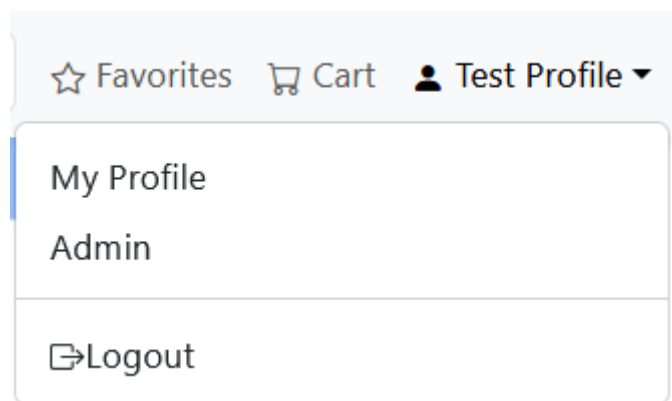
6.55. att. Autorizēšanās lapa

Reģistrēšanās lapa – Reģistrēšanās lapā (skat. 6.56. att.) lietotājam būs iespēja izveidot savu profilu aizpildot visus nepieciešamos laukus un pēc tam nospiežot pogu “Register”. Ja visi lauki būs aizpildīti pēc validācijas prasībām lietotājs tiks novirzīts uz Autorizācijas lapu un lietotājam tiks paziņots par veiksmīgu reģistrāciju. Ja dati būs ievadīti nekorekti, tad, tad tiks izvadīts kļūdas paziņojums par nekorektu lauku. Kā arī lietotājam ir iespēja spiest uz iekrāsotā teksta “login” tādējādi tas tiks novirzīts uz Autorizācijas lapu



6.56. att. Reģistrēšanās lapa

Lietotāja sesijas pabeigšana – Lai pabeigtu lietotāja sesiju (skat. 6.57. att.), lietotājam ir jādodas uz navigāciju un jānospiež uz sava vārda, uzvārda, to izdarot parādīsies pašā apakšā poga “Logout”. Nospiežot pogu “Logout” lietotāja sesija tiks pārtraukta un lietotājs atradīsies viesu skatā.



6.57. att. Lietotāja sesijas pabeigšana

Darba ar lietojumprogrammu pabeigšana – Lai pabeigtu darbu ar lietojumprogrammu ir nepieciešams aizvērt pārlūkprogrammu vai pārlūkprogrammas cilni.

6.4. Testa piemērs

Testu piemēru izveide nodrošina iespēju identificēt un novērst kļūdas un nepilnības jau agrīnā izstrādes posmā, tādējādi uzlabojot galaprodukta kvalitāti un lietotāju pieredzi. Tabulā testēšanas pārskats (skat. 6.1. tabulu) ir aprakstīti testa piemēri, kas tika izstrādāti un izpildīti, lai pārbaudītu sistēmu.

6.1. Tabula

1. Testēšanas pārskats

Nr.	Prasības numurs	Prasības nosaukums	Ievaddati/situācija as apraksts	Sagaidāmais rezultāts	Statuss
1.	1.	Jauna lietotāja reģistrēšana	Neievadot nekādu informāciju tiek nospiesta poga "Register"	Tiks izvadīta informācija: "The name field is required." "The surname field is required. " "The password field is required. " "The phone field is required. " "The email field is required. "	Pareizi
2.			Tiek ievadīti burti telefona numura vietā, netiek ievadīts e-pasts un paroles nesakrīt	Tiks izvadīta informācija: "The password confirmation does not match. " "The phone must be a number. " "The email must be a valid email address. "	Pareizi
3.			Visi dati tiek ievadīti korekti un nospiesta poga register	Tiks izvadīts paziņojums "User registered successfully!"	Pareizi
4.	7	Preces dzēšana	Tiek uzspiesta poga delete items	Parādās PopUp ar tekstu "Are you sure?"	Pareizi
5.			Pēc PopUp tiek uzspiests "cancel"	Prece netiek izdzēsta	Pareizi
6.			Pēc PopUp tiek nospiests "delete"	Prece tiek izdzēsta un lietotājam par to tiek paziņots	Pareizi

7.	2.	Autorizēšanās	Tiek ievadīti pareizi dati: epasts un parole	Lieottāja ievadītie dati tiek salīdzināti ar datu bāze esošajiem datiem, tie sakrīt un lietotājs tiek ielaists savā profilā	Pareizi
8.			Lietotājvārda vietā ievada nepareizu e-pastu	Tieks izvadīts paziņojums "Incorrect email or password. Please try again"	Pareizi
9.			Netiek ievadīts nekas un uzspiests login	Tieks izvadīts paziņojums "Incorrect email or password. Please try again"	Pareizi
10.	4.	Preču filtēšana	Tik ievadīta min un max cena	Tiks izvadītas tikai tās preces, kuras ietilpst šajā preču kategorijā	Pareizi
11.			Ievadīti filtri, kuri nesakrīt ar nevienu preci	Tiks izvadīts paziņojums, ka prece ar šādiem kritērijiem netika atrasta	Pareizi
12.			Tiek noņemti visi filtri	Lietotājam tiks izvadītas visas preces	Pareizi

Visi testa piemēri tika izpildīti, pārlicinoties par sistēmas darbības pareizību un atbilstību noteiktajām prasībām. Testēšanas laikā tika atkārtoti pārbaudīts, vai viss strādā, kā tas bija ieplānots sistēmas projektēšanas posmā.

NOBEIGUMS

Kvalifikācijas darba izstrādes laikā tika izveidota “Elektropreču internetveikala datu uzskaites sistēma”, kas nodrošina iespēju jebkuram lietotājam, neatkarīgi no viņa tehniskās sagatavotības, viegli un ērti pārvaldīt veikala datus. Sistēmas backend izstrādei tika izmantots PHP ar Laravel ietvaru un Axios, savukārt frontend izstrādei tika izvēlēts Vue.js, Vite.js un Bootstrap. Datu glabāšanai tika izmantota MySQL datubāze ar phpMyAdmin, un sistēma tika izvietota XAMPP serverī, izmantojot Apache. Sistēmas funkcionalitāte ietver preču, kategoriju, ražotāju pievienošanu, labošanu, dzēšanu, pasūtījumu veikšanu, preču meklēšanu, filtrēšanu, salīdzināšanu un informācijas izvadu uz PDF un Excel, kā arī noliktavas pārvaldīšanas iespējas. Tika izveidota arī lietotāju reģistrācija un autorizācija.

Pašlaik sistēma tiek izmantota personiskām vajadzībām, bet to var izmantot jebkurš, kuram ir vēlme, jo sistēma ir opensource. Sistēmu noteikti varēs attīstīt uz kaut ko lielāku. Uz doto brīdi ir plāns uzlabot sistēmas struktūru tādējādi samazinot projekta apjomu nesamazinot funkcionalitāti, kā arī ir plāns piesaistīt dizaineri tādējādi uzlabojot UI/UX. Sistēmai arī ir paredzētas backend izmaiņas uzlabojot arhitektūru, tādējādi samazinot API pieprasījumu skaitu un uzlabojot sistēmas ātrdarbību.

INFORMĀCIJAS AVOTI

1. [ANG] Axios dokumentācija - <https://axios-http.com/docs/intro> - (Resurss apskatīts 04.12.2023.).
2. [ANG] Bootstrap dokumentācija - <https://getbootstrap.com/> - (Resurss apskatīts 28.11.2023.).
3. [ANG] ChartJS dokumentācija - <https://www.chartjs.org/docs/latest/> - (Resurss apskatīts 25.05.2024).
4. [ANG] Draw.io dokumentācija - <https://www.drawio.com/doc/> - (Resurss apskatīts 30.05.2024).
5. [ANG] Git dokumentācija - <https://git-scm.com/docs/gitignore> - (Resurss apskatīts 23.11.2023.).
6. [ANG] Git dokumentācija - <https://www.atlassian.com/git/tutorials/using-branches/merge-conflicts> - (Resurss apskatīts 30.05.2024).
7. [ANG] jsPDF dokumentācija - <https://artskydj.github.io/jsPDF/docs/jsPDF.html> - (Resurss apskatīts 26.05.2024).
8. [ANG] Laravel dokumentācijā - <https://laravel.com/docs/10.x> - (Resurss apskatīts 06.11.2023.).
9. [ANG] MySql dokumentācija - <https://dev.mysql.com/doc/> - (Resurss apskatīts 29.05.2024).
10. [ANG] PhpMyAdmin dokumentācija - <https://www.phpmyadmin.net/docs/> - (Resurss apskatīts 29.05.2024).
11. [ANG] Vite.js dokumentācijā - <https://vitejs.dev/> - (Resurss apskatīts 04.12.2023.).
12. [ANG] Vue.js dokumentācijā - <https://vuejs.org/guide/introduction.html> - (Resurss apskatīts 06.11.2023.).
13. [ANG] W3schools dokumentācija - <https://www.w3schools.com/> - (Resurss apskatīts 24.05.2024).
14. [ANG] XAMP dokumentācija - <https://www.apachefriends.org/> - (Resurss apskatīts 29.05.2024).
15. [ANG] xlsx dokumentācija - <https://www.npmjs.com/package/xlsx> - (Resurss apskatīts 24.05.2024).

PIELIKUMI

Pirkuma PDF izdruka



Purchased At: Invalid Date Invalid Date

Buyer: Test Profile
 Email: test@gmail.com
 Phone: 25966512

Item Name	Quantity	Price without VAT (€)	VAT Amount (€)	VAT %	Total Price (€)
HyperX QuadCast S Black	1	214.87	45.12	21.00	259.99
HyperX Cloud Alpha Black/ Red	1	57.02	11.98	21.00	69

Description	Amount (€)
Total Price without VAT	271.89
Total VAT Amount	57.10
Total Price	328.99

Payment Terms and Conditions

Payment is due within 30 days from the purchase date. After 30 days purchases will be canceled!

Issuer:

Name Surname: _____
 Date: _____
 Signature: _____

Receiver:

Name Surname: _____
 Date: _____
 Signature: _____

Preces specifikācijas XLS izdruka

Item ID	Item Name	Keyboard type	Keyboard language	Color	Built-in USB ports	Connection type	Lighting	Number pad	Warranty
38	Steelseries Apex 3 TKL (US)	Gaming	EN (US)	Black	No	USB	No	No	2 Years
39	Razer BlackWidow V3 Tenkeyless Green	Gaming	EN, RU	Black	No	USB	Yes	No	2 Years
40	Razer Huntsman Mini Red Switch (EN)	Gaming	EN (US)	Black	No	USB	Yes	No	2 Years
41	Logitech G213 (US)	Gaming	ES (US)	Black	No	USB	Yes	Yes	2 Years
42	Logitech G413 TKL SE (US)	Gaming	EN (US)	Black	No	USB	No	No	2 Years
43	HyperX "Alloy Origins Blue Switches" (EN)	Gaming	EN (US)	Black	No	USB	Yes	Yes	2 Years
44	Apple "Magic Keyboard with Touch ID and Numeric Keypad" (RU)	Classic	EN, RU	White	No	Bluetooth	No	Yes	1 Year
45	Apple "Magic keyboard" (Int EN)	Classic	EN	White	No	Bluetooth	No	No	1 Year

2. pielikums.

Programmas pirmteksts

AuthController.php

```
<?php

namespace App\Http\Controllers\Api;

use App\Models\User;
use Illuminate\Http\Request;
use App\Http\Controllers\Controller;
use Illuminate\Support\Facades\Auth;
use Illuminate\Support\Facades\Validator;
use Illuminate\Validation\Rule;
use App\Models\FavoriteItems;
use App\Models\Cart;

class AuthController extends Controller
{
    public function register(Request $request)
    {
        $validated = $request->validate([
            'name' => 'required|regex:/^[A-Za-z]+$/',
            'surname' => 'required|regex:/^[A-Za-z]+$/',
            'password' => 'required|string|max:255|confirmed',
            'phone' => 'required|numeric|digits_between:4,16|unique:users',
            'email' => 'required|email|unique:users',
            'admin' => 'boolean',
        ]);

        $validated['password'] = bcrypt($validated['password']);

        $user = User::create($validated);

        return response()->json([
            'status' => 200,
            'message' => 'User registered successfully!',
        ], 200);
    }

    public function login(Request $request)
    {
        $validator = Validator::make($request->all(), [
            'email' => 'required|email',
            'password' => 'required',
        ]);

        if (Auth::attempt(['email' => $request->email, 'password' => $request->password])) {
            $user = auth()->user();
        }
    }
}
```

```

        $token = $user->createToken('auth_token')->plainTextToken;

        return response()->json([
            'message' => 'User logged in successfully!',
            'access_token' => $token,
            'user' => $user,
        ], 200);
    } else {
        return response()->json([
            'message' => 'Invalid credentials!',
        ], 401);
    }
}

public function logout()
{
    auth()->user()->tokens()->delete();
    return response()->json([
        'message' => 'Logged out',
    ]);
}

public function user()
{
    return auth()->user();
}

public function update(Request $request, $id)
{
    $user = User::findOrFail($id);

    $validator = Validator::make($request->all(), [
        'name' => 'required|string|max:255',
        'surname' => 'required|string|max:255',
        'phone' => 'required|numeric|digits_between:4,16',
        'email' => [
            'required',
            'string',
            'email',
            'max:255',
            Rule::unique('users')->ignore($user->id),
        ],
        'admin' => 'boolean',
    ]);

    if ($validator->fails()) {
        return response()->json([
            'status' => 422,
            'message' => 'Validation failed',
        ], 422);
    }

    $user->update($request->all());
    return response()->json($user, 200);
}

```

```

        'errors' => $validator->errors(),
    ], 422);
}

$user->update($request->all());

return response()->json([
    'status' => 200,
    'message' => 'User updated successfully!',
], 200);
}

public function changePassword(Request $request, int $id)
{
    $user = User::find($id);

    if (!$user) {
        return response()->json([
            'status' => 404,
            'message' => 'User not found!',
        ], 404);
    }

    $validator = Validator::make($request->all(), [
        'current_password' => 'required',
        'password' => 'required|string|max:255|confirmed',
    ]);

    if ($validator->fails()) {
        return response()->json([
            'status' => 422,
            'message' => 'Bad Request data wrong!',
            'errors' => $validator->messages(),
        ], 422);
    }

    if (password_verify($request->current_password, $user->password)) {
        $user->update([
            'password' => bcrypt($request->password),
        ]);

        return response()->json([
            'status' => 200,
            'message' => 'Password updated successfully!',
        ], 200);
    } else {
        return response()->json([
            'status' => 422,
            'message' => 'Current password is incorrect!',
        ], 422);
    }
}

```

```

    }
}

public function userAmount()
{
    $userCount = User::count();

    return response()->json([
        'status' => 200,
        'message' => 'Total number of users retrieved successfully!',
        'user_count' => $userCount,
    ], 200);
}

public function getUserCounts($userId)
{
    $favoritesCount = FavoriteItems::where('user_id', $userId)->count();
    $cartCount = Cart::where('user_id', $userId)->count();

    return response()->json([
        'status' => 200,
        'favoritesCount' => $favoritesCount,
        'cartCount' => $cartCount,
    ], 200);
}
}

```

BrandsController.php

```

<?php

namespace App\Http\Controllers\Api;

use App\Models\Brands;
use Illuminate\Http\Request;
use App\Http\Controllers\Controller;
use Illuminate\Support\Facades\Validator;
use Illuminate\Support\Str;

class BrandsController extends Controller
{
    public function index()
    {
        $brands = Brands::withCount('items')->get();
        if ($brands->count() > 0) {
            return response()->json([
                'status' => 200,
                'brands' => $brands,
            ], 200);
        } else {
            return response()->json([

```



```

        'status' => 404,
        'message' => 'No data found!',
    ], 404);
    }
}

public function store(Request $request)
{
    $validator = Validator::make($request->all(), [
        'name' => 'required|string|max:40|unique:brands,name',
        'img' => 'required|image|mimes:jpeg,png,jpg,gif,svg',
    ]);

    if ($validator->fails()) {
        return response()->json([
            'status' => 422,
            'message' => 'Bad Request!',
            'errors' => $validator->messages(),
        ], 422);
    } else {
        $randomString = Str::random(10);
        $imgPath = $request->file('img')->storeAs('uploads', $randomString
        . '.' . $request->file('img')->getClientOriginalExtension(), 'public');

        $brands = Brands::create([
            'name' => $request->name,
            'img' => $randomString . '.' . $request->file('img')-
            >getClientOriginalExtension(),
        ]);

        if ($brands) {
            return response()->json([
                'status' => 200,
                'message' => 'Data created successfully!',
            ], 200);
        } else {
            return response()->json([
                'status' => 500,
                'message' => 'Internal server error!',
            ], 500);
        }
    }
}

public function show($id)
{
    $brands = Brands::find($id);
    if ($brands) {
        return response()->json([
            'status' => 200,

```

```

        'brands' => $brands,
    ], 200);
} else {
    return response()->json([
        'status' => 404,
        'message' => 'No listing found',
    ], 404);
}
}

public function edit($id)
{
    $brands = Brands::find($id);
    if ($brands) {
        return response()->json([
            'status' => 200,
            'brands' => $brands,
        ], 200);
    } else {
        return response()->json([
            'status' => 404,
            'message' => 'No listing found',
        ], 404);
    }
}

public function update(Request $request, int $id)
{
    $brands = Brands::find($id);

    if ($brands) {

        $validator = Validator::make($request->all(), [
            'name' => 'required|string|max:40',
            'img' => $request->hasFile('img') ?
'image|mimes:jpeg,png,jpg,gif,svg' : 'nullable',
        ]);

        if ($validator->fails()) {
            return response()->json([
                'status' => 422,
                'message' => 'Bad Request data wrong!',
                'errors' => $validator->messages(),
            ], 422);
        } else {
            if ($request->hasFile('img')) {
                $randomString = Str::random(10);
                $imgPath = $request->file('img')->storeAs('uploads',
$randomString . '.' . $request->file('img')->getClientOriginalExtension(),
'public');

```

```

        if ($brands->img) {
            $oldImagePath =
public_path("/storage/uploads/{$brands->img}");
            if (file_exists($oldImagePath)) {
                unlink($oldImagePath);
            }
        }

        $brands->update([
            'name' => $request->name,
            'img' => $randomString . '.' . $request->file('img')->
getClientOriginalExtension(),
        ]);
    } else {
        $brands->update([
            'name' => $request->name,
        ]);
    }

    return response()->json([
        'status' => 200,
        'message' => 'Data updated successfully!',
    ], 200);
}
} else {
    return response()->json([
        'status' => 404,
        'message' => 'No listing found',
    ], 404);
}
}

public function destroy($id)
{
    $brand = Brands::find($id);

    if (!$brand) {
        return response()->json([
            'status' => 404,
            'message' => 'Brand not found',
        ], 404);
    }

    $imagePath = public_path("/storage/uploads/{$brand->img}");

    if (file_exists($imagePath)) {
        unlink($imagePath);
    }
}

```

```

        $brand->delete();

        return response()->json([
            'status' => 200,
            'message' => 'Brand deleted successfully',
        ], 200);
    }

}

CartController.php

```

```

<?php
namespace App\Http\Controllers\Api;

use App\Models\Cart;
use Illuminate\Http\Request;
use App\Http\Controllers\Controller;
use Illuminate\Support\Facades\Validator;
use App\Models\Items;

class CartController extends Controller
{
    public function index()
    {
        $cart = Cart::all();
        if ($cart->count() > 0) {
            return response()->json([
                'status' => 200,
                'cart' => $cart,
            ], 200);
        } else {
            return response()->json([
                'status' => 404,
                'message' => 'No data found!',
            ], 404);
        }
    }

    public function store(Request $request)
    {
        $validator = Validator::make($request->all(), [
            'user_id' => 'required|exists:users,id',
            'item_id' => 'required|exists:items,id',
        ]);

        if ($validator->fails()) {
            return response()->json([
                'status' => 422,
                'errors' => $validator->messages(),
            ], 422);
        }
    }
}

```

```

    $existingCart = Cart::where('user_id', $request->user_id)
        ->where('item_id', $request->item_id)
        ->first();

    if ($existingCart) {
        return response()->json([
            'status' => 409,
            'message' => 'Item already in cart',
        ], 409);
    }

    $cart = new Cart();
    $cart->user_id = $request->user_id;
    $cart->item_id = $request->item_id;
    $cart->save();

    return response()->json(['message' => 'Item added to cart
successfully'], 200);
}

public function show($id)
{
    $cart = Cart::find($id);
    if ($cart) {
        return response()->json([
            'status' => 200,
            'cart' => $cart,
        ], 200);
    } else {
        return response()->json([
            'status' => 404,
            'message' => 'No item found',
        ], 404);
    }
}

public function getUserCart($userId)
{
    $cartItems = Cart::with('item')->where('user_id', $userId)->get();
    if ($cartItems->isEmpty()) {
        return response()->json([
            'status' => 404,
            'message' => 'No items found in the cart',
        ], 404);
    }
    return response()->json([
        'status' => 200,
        'cartItems' => $cartItems,
    ], 200);
}

```

```

    }

    public function destroyByCartId(Request $request)
    {
        try {
            $userId = $request->route('user_id');
            $itemId = $request->route('item_id');

            $favorite = Cart::where('user_id', $userId)
                ->where('item_id', $itemId)
                ->first();

            if ($favorite) {
                $favorite->delete();
                return response()->json(['message' => 'Favorite item removed
successfully'], 200);
            } else {
                return response()->json(['message' => 'Favorite item not
found'], 404);
            }
        } catch (\Exception $e) {
            return response()->json(['message' => 'Server error while removing
favorite item', 'error' => $e->getMessage()], 500);
        }
    }

    public function clearCartByUserId($userId)
    {
        $cartItems = Cart::where('user_id', $userId)->delete();

        if ($cartItems) {
            return response()->json(['status' => 200, 'message' => 'Cart
cleared successfully'], 200);
        } else {
            return response()->json(['status' => 404, 'message' => 'No items
found in cart'], 404);
        }
    }

    public function getUserCartCount($userId)
    {
        $cartCount = Cart::where('user_id', $userId)->count();

        return response()->json([
            'status' => 200,
            'count' => $cartCount,
        ], 200);
    }

```

```

    public function isItemInCart($userId, $itemId)
    {
        $isInCart = Cart::where('user_id', $userId)->where('item_id',
$itemId)->exists();
        return response()->json([
            'status' => 200,
            'isInCart' => $isInCart,
        ], 200);
    }

    public function getCartStatus(Request $request)
    {
        $userId = $request->input('userId');
        $itemIds = $request->input('itemIds');

        $cartItems = Cart::where('user_id', $userId)
            ->whereIn('item_id', $itemIds)
            ->pluck('item_id')
            ->toArray();

        $cartStatus = array_fill_keys($itemIds, false);
        foreach ($cartItems as $itemId) {
            $cartStatus[$itemId] = true;
        }

        return response()->json($cartStatus);
    }
}

```

Categoriescontroller.php

```

<?php

namespace App\Http\Controllers\Api;

use App\Models\Categories;
use Illuminate\Http\Request;
use App\Http\Controllers\Controller;
use Illuminate\Support\Facades\Validator;
use App\Models\SpecificationTitle;

class CategoriesController extends Controller
{
    public function index()
    {
        $categories = Categories::withCount('items')->get();
        if ($categories->count() > 0) {
            return response()->json([
                'status' => 200,
                'categories' => $categories,
            ], 200);
        }
    }
}

```

```

    } else {
        return response()->json([
            'status' => 404,
            'message' => 'No data found!',
        ], 404);
    }
}

public function store(Request $request)
{
    $validator = Validator::make($request->all(), [
        'category_name' =>
        'required|string|max:40|unique:categories,category_name',
        'specification_titles' => 'sometimes|array',
        'specification_titles.*.title' =>
        'required_with:specification_titles|string|max:255',
    ]);

    if ($validator->fails()) {
        return response()->json([
            'status' => 422,
            'message' => 'Bad Request!',
            'errors' => $validator->messages(),
        ], 422);
    }

    if ($request->has('specification_titles')) {
        $titles = array_column($request->specification_titles, 'title');
        if (count($titles) !== count(array_unique($titles))) {
            return response()->json([
                'status' => 422,
                'message' => 'Each specification title must be unique
within a category.',
                'errors' => ['specification_titles' => ['Each
specification title must be unique within a category.']],
            ], 422);
        }
    }

    $category = Categories::create([
        'category_name' => $request->category_name,
    ]);

    if ($request->has('specification_titles')) {
        foreach ($request->specification_titles as $specTitle) {
            SpecificationTitle::create([
                'category_id' => $category->id,
                'specification_title' => $specTitle['title'],
            ]);
        }
    }
}

```



```

    }

    return response()->json([
        'status' => 200,
        'message' => 'Data created successfully!',
    ], 200);
}

public function show($id)
{
    $categories = Categories::find($id);
    if ($categories) {
        return response()->json([
            'status' => 200,
            'categories' => $categories,
        ], 200);
    } else {
        return response()->json([
            'status' => 404,
            'message' => 'No listing found',
        ], 404);
    }
}

public function getCategories()
{
    $categories = Categories::all();
    return response()->json(['categories' => $categories], 200);
}

public function edit($id)
{
    $categories = Categories::find($id);
    if ($categories) {
        return response()->json([
            'status' => 200,
            'categories' => $categories,
        ], 200);
    } else {
        return response()->json([
            'status' => 404,
            'message' => 'No listing found',
        ], 404);
    }
}

public function update(Request $request, int $id)
{
    $categories = Categories::find($id);
    if ($categories) {

```

```

$validator = Validator::make($request->all(), [
    'category_name' => 'required|string|max:40',
]);

if ($validator->fails()) {
    return response()->json([
        'status' => 422,
        'message' => 'Bad Request data wrong!',
        'errors' => $validator->messages(),
    ], 422);
} else {
    $categories->update([
        'category_name' => $request->category_name,
    ]);

    if ($categories) {
        return response()->json([
            'status' => 200,
            'message' => 'Data updated successfully!',
        ], 200);
    } else {
        return response()->json([
            'status' => 404,
            'message' => 'No listing found!',
        ], 404);
    }
}
} else {
    return response()->json([
        'status' => 404,
        'message' => 'No listing found',
    ], 404);
}
}

public function destroy($id)
{
    $categories = Categories::find($id);
    if ($categories) {
        $categories->delete();
        return response()->json([
            'status' => 200,
            'message' => 'Data deleted successfully!',
        ], 200);
    } else {
        return response()->json([
            'status' => 404,
            'message' => 'No listing found',
        ], 404);
    }
}

```

```
}  
}
```

FavoriteItemsController.php

```
<?php
```

```
namespace App\Http\Controllers\Api;  
  
use App\Models\FavoriteItems;  
use Illuminate\Http\Request;  
use App\Http\Controllers\Controller;  
use Illuminate\Support\Facades\Validator;  
  
class FavoriteItemsController extends Controller  
{  
    public function index()  
    {  
        $favorites = FavoriteItems::all();  
        if ($favorites->count() > 0) {  
            return response()->json([  
                'status' => 200,  
                'favorites' => $favorites,  
            ], 200);  
        } else {  
            return response()->json([  
                'status' => 404,  
                'message' => 'No data found!',  
            ], 404);  
        }  
    }  
  
    public function userFavorites($userId)  
    {  
        try {  
            $favorites = FavoriteItems::where('user_id', $userId)  
                ->with('item')  
                ->get();  
  
            if ($favorites->isEmpty()) {  
                return response()->json([  
                    'status' => 404,  
                    'message' => 'No favorites found for this user.',  
                ], 404);  
            }  
  
            return response()->json([  
                'status' => 200,  
                'favorites' => $favorites,  
            ], 200);  
        } catch (\Exception $e) {  
            return response()->json([
```

```

        'status' => 500,
        'message' => 'Server error while fetching favorite items.',
        'error' => $e->getMessage(),
    ], 500);
}
}

public function store(Request $request)
{
    $validator = Validator::make($request->all(), [
        'user_id' => 'required|exists:users,id',
        'item_id' => 'required|exists:items,id',
    ]);

    if ($validator->fails()) {
        return response()->json([
            'status' => 422,
            'errors' => $validator->messages(),
        ], 422);
    }

    $existingFavorite = FavoriteItems::where('user_id', $request->user_id)
        ->where('item_id', $request->item_id)
        ->first();

    if ($existingFavorite) {
        return response()->json([
            'status' => 409,
            'message' => 'Item already in favorites',
        ], 409);
    }

    $favoriteItem = new FavoriteItems();
    $favoriteItem->user_id = $request->user_id;
    $favoriteItem->item_id = $request->item_id;
    $favoriteItem->save();

    return response()->json(['message' => 'Favorite item saved
successfully'], 200);
}

public function show($id)
{
    $favorite = FavoriteItems::find($id);
    if ($favorite) {
        return response()->json([
            'status' => 200,
            'favorite' => $favorite,
        ], 200);
    } else {

```

```

        return response()->json([
            'status' => 404,
            'message' => 'No listing found',
        ], 404);
    }
}

public function userFavoritesCount($userId)
{
    $count = FavoriteItems::where('user_id', $userId)->count();
    return response()->json([
        'status' => 200,
        'count' => $count,
    ], 200);
}

public function isFavorite($userId, $itemId)
{
    $isFavorite = FavoriteItems::where('user_id', $userId)-
>where('item_id', $itemId)->exists();
    return response()->json([
        'status' => 200,
        'isFavorite' => $isFavorite,
    ], 200);
}

public function destroyByItemId(Request $request)
{
    try {
        $userId = $request->route('user_id');
        $itemId = $request->route('item_id');

        $favorite = FavoriteItems::where('user_id', $userId)
            ->where('item_id', $itemId)
            ->first();

        if ($favorite) {
            $favorite->delete();
            return response()->json(['message' => 'Favorite item removed
successfully'], 200);
        } else {
            return response()->json(['message' => 'Favorite item not
found'], 404);
        }
    } catch (\Exception $e) {
        return response()->json(['message' => 'Server error while removing
favorite item', 'error' => $e->getMessage()], 500);
    }
}
}

```

```

    public function clearUserFavorites($userId)
    {
        try {
            FavoriteItems::where('user_id', $userId)->delete();
            return response()->json(['message' => 'All favorite items removed
successfully'], 200);
        } catch (\Exception $e) {
            return response()->json(['message' => 'Server error while removing
favorite items', 'error' => $e->getMessage()], 500);
        }
    }

    public function getFavoritesStatus(Request $request)
    {
        $userId = $request->input('userId');
        $itemIds = $request->input('itemIds');

        $favorites = FavoriteItems::where('user_id', $userId)
            ->whereIn('item_id', $itemIds)
            ->pluck('item_id')
            ->toArray();

        $favoritesStatus = array_fill_keys($itemIds, false);
        foreach ($favorites as $itemId) {
            $favoritesStatus[$itemId] = true;
        }

        return response()->json($favoritesStatus);
    }
}

```

FinancesController.php

```

<?php

namespace App\Http\Controllers\Api;

use App\Models\Purchase;
use App\Models\Items;
use Illuminate\Http\Request;
use App\Http\Controllers\Controller;

class FinancesController extends Controller
{
    public function getFinances(Request $request)
    {
        $purchases = Purchase::with(['item', 'item.category'])
            ->where('status', 'closed')
            ->get();
    }
}

```

```

    $categoryFinances = $purchases-
>groupBy('item.category.category_name')->map(function ($categoryPurchases) {
    $category_id = $categoryPurchases->first()->item->category->id;
    $totalEarnedWithVat = $categoryPurchases->sum('total_price');
    $totalVat = $categoryPurchases->sum(function ($purchase) {
        return $purchase->total_price * ($purchase->vat / (100 +
$purchase->vat));
    });
    $totalEarnedWithoutVat = $totalEarnedWithVat - $totalVat;

    return [
        'category_id' => $category_id,
        'total_earned_with_vat' => $totalEarnedWithVat,
        'total_vat' => $totalVat,
        'total_earned_without_vat' => $totalEarnedWithoutVat
    ];
});

    $itemFinances = $purchases->groupBy('item.name')->map(function
($itemPurchases) {
    $item_id = $itemPurchases->first()->item->id;
    $totalEarnedWithVat = $itemPurchases->sum('total_price');
    $totalVat = $itemPurchases->sum(function ($purchase) {
        return $purchase->total_price * ($purchase->vat / (100 +
$purchase->vat));
    });
    $totalEarnedWithoutVat = $totalEarnedWithVat - $totalVat;

    return [
        'item_id' => $item_id,
        'total_earned_with_vat' => $totalEarnedWithVat,
        'total_vat' => $totalVat,
        'total_earned_without_vat' => $totalEarnedWithoutVat
    ];
});

    $mostSoldItem = Items::orderBy('sold', 'desc')->first();

    $activeOrders = Purchase::where('status', 'active')->count();
    $canceledOrders = Purchase::where('status', 'canceled')->count();
    $closedOrders = Purchase::where('status', 'closed')->count();

    $totalEarnedWithVat = $purchases->sum('total_price');
    $totalVat = $purchases->sum(function ($purchase) {
        return $purchase->total_price * ($purchase->vat / (100 +
$purchase->vat));
    });
    $totalEarnedWithoutVat = $totalEarnedWithVat - $totalVat;

    return response()->json([

```

```

        'categories' => $categoryFinances,
        'items' => $itemFinances,
        'mostSoldItem' => [
            'name' => $mostSoldItem->name,
            'description' => $mostSoldItem->description,
            'price' => $mostSoldItem->price,
            'image' => $mostSoldItem->img
        ],
        'activeOrders' => $activeOrders,
        'canceledOrders' => $canceledOrders,
        'closedOrders' => $closedOrders,
        'totalEarnedWithVat' => $totalEarnedWithVat,
        'totalVat' => $totalVat,
        'totalEarnedWithoutVat' => $totalEarnedWithoutVat
    ]);
}
}
}

```

ItemsController.php

```

<?php

namespace App\Http\Controllers\Api;

use App\Models\Items;
use Illuminate\Http\Request;
use App\Http\Controllers\Controller;
use Illuminate\Support\Facades\Validator;
use Illuminate\Support\Str;
use App\Models\SpecificationTitle;
use App\Models\SpecificationDescription;

class ItemsController extends Controller
{
    public function index()
    {
        $items = Items::all();
        if ($items->count() > 0) {
            return response()->json([
                'status' => 200,
                'items' => $items,
            ], 200);
        } else {
            return response()->json([
                'status' => 404,
                'message' => 'No data found!',
            ], 404);
        }
    }

    public function store(Request $request)

```



```

{
    $validator = Validator::make($request->all(), [
        'name' => 'required|string',
        'description' => 'required|string|max:1000',
        'price' => 'required|numeric',
        'brand_id' => 'required|exists:brands,id',
        'categories_id' => 'required|exists:categories,id',
        'img' => 'required|image|mimes:jpeg,png,jpg,gif,svg',
        'specifications.*.specification_title_id' =>
'required_with:specifications|exists:specification_titles,id',
        'specifications.*.description' =>
'required_with:specifications|string',
    ]);

    if ($validator->fails()) {
        return response()->json([
            'status' => 422,
            'message' => 'Bad Request!',
            'errors' => $validator->messages(),
        ], 422);
    }

    $randomString = Str::random(10);
    $file = $request->file('img');
    $fileName = $randomString . '.' . $file->getClientOriginalExtension();
    $file->storeAs('uploads', $fileName, 'public');

    $item = Items::create([
        'name' => $request->name,
        'description' => $request->description,
        'price' => $request->price,
        'brand_id' => $request->brand_id,
        'categories_id' => $request->categories_id,
        'img' => $fileName,
    ]);

    if ($request->has('specifications')) {
        $specifications = json_decode($request->specifications, true);
        foreach ($specifications as $specification) {
            SpecificationDescription::create([
                'item_id' => $item->id,
                'specification_title_id' =>
$specification['specification_title_id'],
                'description' => $specification['description'],
            ]);
        }
    }

    return response()->json([
        'status' => 200,
    ]);
}

```

```

        'message' => 'Data created successfully!',
    ], 200);
}

public function show($id)
{
    $items = Items::find($id);
    if ($items) {
        return response()->json([
            'status' => 200,
            'items' => $items,
        ], 200);
    } else {
        return response()->json([
            'status' => 404,
            'message' => 'No listing found',
        ], 404);
    }
}

public function edit($id)
{
    $item = Items::find($id);
    if ($item) {
        return response()->json([
            'status' => 200,
            'items' => $item,
        ], 200);
    } else {
        return response()->json([
            'status' => 404,
            'message' => 'No listing found',
        ], 404);
    }
}

public function update(Request $request, int $id)
{
    $item = Items::find($id);

    if ($item) {
        $validator = Validator::make($request->all(), [
            'name' => 'required|string',
            'description' => 'required|string|max:1000',
            'price' => 'required|numeric',
            'brand_id' => 'required|exists:brands,id',
            'categories_id' => 'required|exists:categories,id',
            'img' => $request->hasFile('img') ?
            'image|mimes:jpeg,png,jpg,gif,svg' : 'nullable',
        ]);
    }
}

```

```

        if ($validator->fails()) {
            return response()->json([
                'status' => 422,
                'message' => 'Bad Request data wrong!',
                'errors' => $validator->messages(),
            ], 422);
        } else {
            if ($request->hasFile('img')) {
                $randomString = Str::random(10);
                $imgPath = $request->file('img')->storeAs('uploads',
$randomString . '.' . $request->file('img')->getClientOriginalExtension(),
'public');

                if ($item->img) {
                    $oldImagePath = public_path("/storage/uploads/{$item-
>img}");

                    if (file_exists($oldImagePath)) {
                        unlink($oldImagePath);
                    }
                }

                $item->update([
                    'name' => $request->name,
                    'description' => $request->description,
                    'price' => $request->price,
                    'brand_id' => $request->brand_id,
                    'categories_id' => $request->categories_id,
                    'img' => $randomString . '.' . $request->file('img')-
>getClientOriginalExtension(),
                ]);
            } else {
                $item->update([
                    'name' => $request->name,
                    'description' => $request->description,
                    'price' => $request->price,
                    'brand_id' => $request->brand_id,
                    'categories_id' => $request->categories_id,
                ]);
            }

            return response()->json([
                'status' => 200,
                'message' => 'Data updated successfully!',
            ], 200);
        }
    } else {
        return response()->json([
            'status' => 404,
            'message' => 'No listing found',
        ], 404);
    }
}

```

```

        ], 404);
    }
}

public function destroy($id)
{
    $item = Items::find($id);

    if (!$item) {
        return response()->json([
            'status' => 404,
            'message' => 'Item not found',
        ], 404);
    }

    $imagePath = public_path("/storage/uploads/{$item->img}");

    if (file_exists($imagePath)) {
        unlink($imagePath);
    }

    $item->delete();

    return response()->json([
        'status' => 200,
        'message' => 'Item deleted successfully',
    ], 200);
}

public function frontPageItems()
{
    $items = Items::latest()->take(4)->get();

    if ($items->count() > 0) {
        return response()->json([
            'status' => 200,
            'items' => $items,
        ], 200);
    } else {
        return response()->json([
            'status' => 404,
            'message' => 'No data found!',
        ], 404);
    }
}

public function searchItems(Request $request)
{
    $name = $request->query('name');
    $items = Items::where('name', 'LIKE', "%$name%")->get();
}

```

```

        return response()->json(['items' => $items]);
    }

    public function getCategoryItemCount()
    {
        $categoryCounts = Items::selectRaw('categories_id, COUNT(*) as
item_count')
            ->groupBy('categories_id')
            ->with('category')
            ->get();

        $data = $categoryCounts->map(function ($item) {
            return [
                'category_name' => $item->category->category_name,
                'item_count' => $item->item_count,
            ];
        });

        return response()->json(['category_counts' => $data]);
    }

    public function getItemsByBrand($brandId)
    {
        $items = Items::where('brand_id', $brandId)->get();

        if ($items->count() > 0) {
            return response()->json([
                'status' => 200,
                'items' => $items,
            ], 200);
        } else {
            return response()->json([
                'status' => 404,
                'message' => 'No items found for this brand!',
            ], 404);
        }
    }

    public function getSimilarItems($id)
    {
        $currentItem = Items::find($id);

        if (!$currentItem) {
            return response()->json([
                'status' => 404,
                'message' => 'Item not found',
            ], 404);
        }
    }

```

```

        $similarItems = Items::where('categories_id', $currentItem-
>categories_id)
            ->where('id', '!=', $id)
            ->take(4)
            ->get();

        if ($similarItems->count() > 0) {
            return response()->json([
                'status' => 200,
                'items' => $similarItems,
            ], 200);
        } else {
            return response()->json([
                'status' => 404,
                'message' => 'No similar items found',
            ], 404);
        }
    }
}

public function updateInventory(Request $request, int $id)
{
    $item = Items::find($id);

    if ($item) {
        $validator = Validator::make($request->all(), [
            'price' => 'required|numeric',
            'amount' => 'required|integer',
            'vat' => 'required|numeric|in:0,5,12,21'
        ]);

        if ($validator->fails()) {
            return response()->json([
                'status' => 422,
                'message' => 'Bad Request data wrong!',
                'errors' => $validator->messages(),
            ], 422);
        } else {
            $item->update([
                'price' => $request->price,
                'amount' => $request->amount,
                'vat' => $request->vat
            ]);

            return response()->json([
                'status' => 200,
                'message' => 'Data updated successfully!',
            ], 200);
        }
    } else {
        return response()->json([

```

```

        'status' => 404,
        'message' => 'No listing found',
    ], 404);
    }
}

public function updateItem(Request $request, $id)
{
    $item = Items::find($id);

    if ($item) {
        $item->sold = $request->input('sold', $item->sold);
        $item->reserved = $request->input('reserved', $item->reserved);
        $item->amount = $request->input('amount', $item->amount);
        $item->save();

        return response()->json(['status' => 'success', 'message' => 'Item
updated successfully']);
    } else {
        return response()->json(['status' => 'error', 'message' => 'Item
not found'], 404);
    }
}

public function getSpecificationTitles($categoryId)
{
    $specificationTitles = SpecificationTitle::where('category_id',
$categoryId)->get();

    if ($specificationTitles->count() > 0) {
        return response()->json([
            'status' => 200,
            'specification_titles' => $specificationTitles,
        ], 200);
    } else {
        return response()->json([
            'status' => 404,
            'message' => 'No specification titles found!',
        ], 404);
    }
}

public function getSpecifications($itemId)
{
    try {
        $specifications = SpecificationDescription::where('item_id',
$itemId)
            ->with('specificationTitle')
            ->get();
    }
}

```

```

        if ($specifications->count() > 0) {
            return response()->json([
                'status' => 200,
                'specifications' => $specifications,
            ], 200);
        } else {
            return response()->json([
                'status' => 404,
                'message' => 'No specifications found!',
            ], 404);
        }
    } catch (\Exception $e) {
        return response()->json([
            'status' => 500,
            'message' => 'Server error',
            'error' => $e->getMessage(),
        ], 500);
    }
}

public function getSpecificationsDescription()
{
    try {
        $specifications = SpecificationDescription::with('item',
'specificationTitle')->get();

        if ($specifications->count() > 0) {
            return response()->json([
                'status' => 200,
                'specifications' => $specifications,
            ], 200);
        } else {
            return response()->json([
                'status' => 404,
                'message' => 'No specifications found!',
            ], 404);
        }
    } catch (\Exception $e) {
        return response()->json([
            'status' => 500,
            'message' => 'Server error',
            'error' => $e->getMessage(),
        ], 500);
    }
}
}
}

```



```

<?php

namespace App\Http\Controllers\Api;

use App\Models\Purchase;
use App\Models\Cart;
use App\Models\Items;
use Illuminate\Http\Request;
use App\Http\Controllers\Controller;
use Illuminate\Support\Facades\Validator;

class PurchaseController extends Controller
{
    public function store(Request $request)
    {
        $validator = Validator::make($request->all(), [
            'items' => 'required|array',
            'items.*.id' => 'required|exists:items,id',
            'items.*.quantity' => 'required|integer|min:1',
        ]);

        if ($validator->fails()) {
            return response()->json([
                'status' => 422,
                'message' => 'Bad Request!',
                'errors' => $validator->messages(),
            ], 422);
        }

        $itemsData = $request->input('items');
        $user = auth()->user();

        if (!$user) {
            return response()->json([
                'status' => 401,
                'message' => 'Unauthorized',
            ], 401);
        }

        $userId = $user->id;

        try {
            foreach ($itemsData as $itemData) {
                $item = Items::find($itemData['id']);
                if ($item->amount < $itemData['quantity']) {
                    return response()->json([
                        'status' => 400,
                        'message' => "Insufficient amount for item {$item-
>name}",

```

```

        ], 400);
    }

    $item->amount -= $itemData['quantity'];
    $item->reserved += $itemData['quantity'];
    $item->save();

    $totalPriceWithVat = $item->price * $itemData['quantity'];

    Purchase::create([
        'user_id' => $userId,
        'item_id' => $item->id,
        'quantity' => $itemData['quantity'],
        'total_price' => $totalPriceWithVat,
        'vat' => $item->vat,
        'status' => 'active',
    ]);
}

Cart::where('user_id', $userId)->delete();

return response()->json([
    'status' => 200,
    'message' => 'Purchase successful!',
], 200);
} catch (\Exception $e) {
    return response()->json([
        'status' => 500,
        'message' => 'An error occurred during the purchase process',
        'error' => $e->getMessage(),
    ], 500);
}
}

public function getUserPurchases($userId)
{
    $purchases = Purchase::with(['item', 'user'])
        ->where('user_id', $userId)
        ->get();

    if ($purchases->isEmpty()) {
        return response()->json([
            'status' => 404,
            'message' => 'No purchases found',
        ], 404);
    }

    return response()->json([
        'status' => 200,
        'purchases' => $purchases,
    ], 200);
}

```

```

    ], 200);
}

public function updateStatus(Request $request, $id)
{
    $status = $request->input('status');
    $purchase = Purchase::find($id);

    if ($purchase) {
        $purchase->status = $status;
        $purchase->save();

        $item = Items::find($purchase->item_id);
        if ($item) {
            $item->sold = $item->sold ?? 0;
            $item->reserved = $item->reserved ?? 0;
            $item->amount = $item->amount ?? 0;

            if ($status === 'closed') {
                $item->amount += $purchase->quantity;
                $item->reserved -= $purchase->quantity;
            } elseif ($status === 'canceled') {
                $item->amount += $purchase->quantity;
                $item->reserved -= $purchase->quantity;
            }

            $item->reserved = max(0, $item->reserved);
            $item->amount = max(0, $item->amount);

            $item->save();

            return response()->json(['status' => 'success', 'message' =>
                'Purchase status updated successfully']);
        } else {
            return response()->json(['status' => 'error', 'message' =>
                'Item not found'], 404);
        }
    } else {
        return response()->json(['status' => 'error', 'message' =>
            'Purchase not found'], 404);
    }
}

public function getTotalSpendingPerDay()
{
    $spendingData = Purchase::selectRaw('DATE(created_at) as date,
SUM(total_price) as total_spent')
        ->where('status', 'closed')
        ->groupBy('date')
        ->orderBy('date', 'ASC')

```

```

        ->get();

        return response()->json([
            'status' => 200,
            'data' => $spendingData,
        ], 200);
    }
}

```

SpecificationTitleController.php

```

<?php

namespace App\Http\Controllers\Api;

use App\Http\Controllers\Controller;
use App\Models\SpecificationTitle;
use Illuminate\Http\Request;

class SpecificationTitleController extends Controller
{
    public function index()
    {
        $specificationTitles = SpecificationTitle::all();
        return response()->json([
            'status' => 200,
            'specification_titles' => $specificationTitles,
        ], 200);
    }

    public function show($id)
    {
        $specificationTitle = SpecificationTitle::findOrFail($id);
        return response()->json([
            'status' => 200,
            'specification_title' => $specificationTitle,
        ]);
    }

    public function store(Request $request)
    {
        $request->validate([
            'category_id' => 'required|exists:categories,id',
            'specification_titles' => 'required|array',
            'specification_titles.*' => 'required|string',
        ]);

        try {
            foreach ($request->specification_titles as $title) {
                SpecificationTitle::create([

```

```

        'category_id' => $request->category_id,
        'specification_title' => $title,
    ]);
}

return response()->json(['message' => 'Specification Titles added
successfully'], 201);
} catch (\Exception $e) {
    [
        'message' => $e->getMessage(),
        'trace' => $e->getTraceAsString(),
    ];

return response()->json(['message' => 'Failed to add specification
titles'], 500);
}

}

public function update(Request $request, $id)
{
    try {
        $request->validate([
            'specification_title' => 'required|string|max:255',
        ]);

        $specificationTitle = SpecificationTitle::findOrFail($id);
        $specificationTitle->update([
            'specification_title' => $request->specification_title,
        ]);

        return response()->json([
            'status' => 200,
            'message' => 'Specification title updated successfully',
        ]);
    } catch (\Exception $e) {
        return response()->json([
            'status' => 500,
            'message' => 'Internal Server Error',
            'error' => $e->getMessage(),
        ], 500);
    }
}

public function destroy($id)
{
    try {
        $specificationTitle = SpecificationTitle::findOrFail($id);
        $specificationTitle->delete();

        return response()->json([

```

```

        'status' => 200,
        'message' => 'Specification title deleted successfully',
    ]);
    } catch (\Exception $e) {
        \Log::error('Error deleting specification title: ' . $e-
>getMessage());
        return response()->json([
            'status' => 500,
            'message' => 'Internal Server Error',
            'error' => $e->getMessage(),
        ], 500);
    }
}
}
}

```

UserController.php

```

<?php

namespace App\Http\Controllers\Api;

use App\Models\User;
use Illuminate\Http\Request;
use App\Http\Controllers\Controller;
use Illuminate\Support\Facades\DB;

class UserController extends Controller
{
    public function getUserRegistrations()
    {
        $registrations = User::select(DB::raw('DATE(created_at) as date'),
DB::raw('count(*) as count'))
            ->groupBy('date')
            ->orderBy('date')
            ->get();

        return response()->json([
            'status' => 200,
            'registrations' => $registrations,
        ]);
    }
}

```

api.php

```

<?php

use Illuminate\Support\Facades\Route;
use App\Http\Controllers\Api\AuthController;
use App\Http\Controllers\Api\BrandsController;
use App\Http\Controllers\Api\ItemsController;

```

```

use App\Http\Controllers\Api\CATEGORIESController;
use App\Http\Controllers\Api\FavoriteItemsController;
use App\Http\Controllers\Api\CartController;
use App\Http\Controllers\Api\PurchaseController;
use App\Http\Controllers\Api\UserController;
use App\Http\Controllers\Api\FinancesController;
use App\Http\Controllers\Api\SpecificationTitleController;

Route::middleware('auth:sanctum')->group(function () {

    // Auth
    Route::get('logout', [AuthController::class, 'logout']);
    Route::get('user', [AuthController::class, 'user']);
    Route::put('profile/edit/{id}', [AuthController::class, 'update']);
    Route::put('profile/change-password/{id}', [AuthController::class,
'changePassword']);

    // Brands
    Route::post('brands', [BrandsController::class, 'store']);
    Route::get('brands/{id}/edit', [BrandsController::class, 'edit']);
    Route::delete('brands/{id}/delete', [BrandsController::class, 'destroy']);

    // Items
    Route::post('items', [ItemsController::class, 'store']);
    Route::get('items/{id}/edit', [ItemsController::class, 'edit']);
    Route::put('items/{id}/edit', [ItemsController::class, 'update']);
    Route::delete('items/{id}/delete', [ItemsController::class, 'destroy']);
    Route::post('/items/purchase', [ItemsController::class, 'purchase']);

    // Categories
    Route::post('categories', [CategoriesController::class, 'store']);
    Route::get('categories/{id}/edit', [CategoriesController::class, 'edit']);
    Route::put('categories/{id}/edit', [CategoriesController::class,
'update']);
    Route::delete('categories/{id}/delete', [CategoriesController::class,
'destroy']);

    // Favorite Items
    Route::post('favorites', [FavoriteItemsController::class, 'store']);
    Route::get('favorites', [FavoriteItemsController::class, 'index']);
    Route::get('favorites/{id}', [FavoriteItemsController::class, 'show']);
    Route::get('favorites/user', [FavoriteItemsController::class,
'getUserFavorites']);

    // Cart
    Route::post('cart', [CartController::class, 'store']);
    Route::get('cart', [CartController::class, 'index']);
    Route::get('cart/{id}', [CartController::class, 'show']);
    Route::get('cart/user', [CartController::class, 'getUserCart']);

```

```

    // Purchase
    Route::post('purchase', [PurchaseController::class, 'store']);
    Route::get('purchases/user/{userId}', [PurchaseController::class,
'getUserPurchases']);
    Route::patch('purchases/{id}/status', [PurchaseController::class,
'updateStatus']);
    Route::get('total-spending-per-day', [PurchaseController::class,
'getTotalSpendingPerDay']);
});

// Auth API
Route::post('register', [AuthController::class, 'register']);
Route::post('login', [AuthController::class, 'login']);

// Brands API
Route::get('brands/{id}', [BrandsController::class, 'show']);
Route::get('brands', [BrandsController::class, 'index']);
Route::get('brands/{brandId}/products', [ItemsController::class,
'getItemsByBrand']);
Route::put('brands/{id}/edit', [BrandsController::class, 'update']);

// Items API
Route::get('items', [ItemsController::class, 'index']);
Route::get('items/{id}', [ItemsController::class, 'show']);
Route::get('/front-page-items', [ItemsController::class, 'frontPageItems']);
Route::get('/items/search', [ItemsController::class, 'searchItems']);
Route::get('items-category-count', [ItemsController::class,
'getCategoryItemCount']);
Route::get('/items/similar/{id}', [ItemsController::class,
'getSimilarItems']);
Route::put('items/{id}/update-inventory', [ItemsController::class,
'updateInventory']);

// Categories API
Route::get('categories', [CategoriesController::class, 'index']);
Route::get('categories/{id}', [CategoriesController::class, 'show']);

// Favorite Items API
Route::get('favorites', [FavoriteItemsController::class, 'index']);
Route::get('user/{id}/favorites-count', [FavoriteItemsController::class,
'userFavoritesCount']);
Route::get('user/{userId}/favorite/{itemId}', [FavoriteItemsController::class,
'isFavorite']);
Route::delete('favorites/item/{item_id}-{user_id}',
[FavoriteItemsController::class, 'destroyByItemId']);
Route::get('favorites/user/{userId}', [FavoriteItemsController::class,
'getUserFavorites']);
Route::get('favorites/user/{userId}', [FavoriteItemsController::class,
'userFavorites']);

```



```

Route::delete('favorites/user/{userId}/clear',
[FavoriteItemsController::class, 'clearUserFavorites']);

// Users API
Route::get('user-amount', [AuthController::class, 'userAmount']);
Route::get('user/{userId}/counts', [AuthController::class, 'getUserCounts']);
Route::post('user/favorites-status', [FavoriteItemsController::class,
'getFavoritesStatus']);
Route::post('user/cart-status', [CartController::class, 'getCartStatus']);
Route::get('/user-registrations', [UserController::class,
'getUserRegistrations']);

// Cart API
Route::get('cart/user/{userId}', [CartController::class, 'getUserCart']);
Route::get('cart/user/{userId}/count', [CartController::class,
'getUserCartCount']);
Route::delete('cart/item/{item_id}-{user_id}', [CartController::class,
'destroyByCartId']);
Route::get('cart/user/{userId}/item/{itemId}', [CartController::class,
'isItemInCart']);
Route::delete('cart/clear/{userId}', [CartController::class,
'clearCartByUserId']);

// Admin purchases API
Route::patch('/items/{id}', [ItemsController::class, 'updateItem']);

// Finance API
Route::get('/finances', [FinancesController::class, 'getFinances']);

// Specifications API
Route::get('/specification_titles/{categoryId}', [ItemsController::class,
'getSpecificationTitles']);
Route::get('/items/{itemId}/specifications', [ItemsController::class,
'getSpecifications']);
Route::get('/specification-titles', [SpecificationTitleController::class,
'index']);
Route::get('/specification-titles/{id}', [SpecificationTitleController::class,
'show']);
Route::put('/specification-titles/{id}', [SpecificationTitleController::class,
'update']);
Route::post('/specification-titles', [SpecificationTitleController::class,
'store']);
Route::delete('/specification-titles/{id}/delete',
[SpecificationTitleController::class, 'destroy']);
Route::get('/specification-descriptions', [ItemsController::class,
'getSpecificationsDescription']);

```

```

import jsPDF from 'jspdf';
import 'jspdf-autotable';

const formatDate = (datetime) => {
  const date = new Date(datetime);
  const formattedDate = date.toLocaleDateString('en-GB').replace(/\\/g,
  '.'); // DD.MM.YYYY
  const formattedTime = date.toLocaleTimeString(); // HH:MM:SS
  return `${formattedDate} ${formattedTime}`;
};

const calculateVAT = (price, vatRate) => {
  const priceWithoutVAT = price / (1 + vatRate / 100);
  const vatAmount = price - priceWithoutVAT;
  return {
    priceWithoutVAT: priceWithoutVAT.toFixed(2),
    vatAmount: vatAmount.toFixed(2),
    vatPercentage: vatRate.toFixed(2)
  };
};

export const generatePDF = async (time, purchaseGroup, user) => {
  const doc = new jsPDF();
  let yPosition = 20;

  const logoUrl = '/favicon.ico';
  const logoBase64 = await fetch(logoUrl)
    .then(res => res.blob())
    .then(blob => new Promise((resolve, reject) => {
      const reader = new FileReader();
      reader.onloadend = () => resolve(reader.result);
      reader.onerror = reject;
      reader.readAsDataURL(blob);
    })));

  const logoHeight = 15;
  const logoWidth = 15;
  const logoX = 80;
  const logoY = 10;
  doc.addImage(logoBase64, 'PNG', logoX, logoY, logoWidth, logoHeight);

  yPosition += 20;

  doc.setFontSize(18);
  doc.text('Frenko', 105, logoY + logoHeight / 2 + 4, null, null, 'center');

  yPosition += 10;

  doc.setFontSize(12);

```

```

    doc.text(`Purchased At: ${formatDateTime(purchaseGroup.createdAt)}`, 10,
yPosition);
    yPosition += 10;

    doc.setFontSize(12);
    doc.text(`Buyer: ${user.name} ${user.surname}`, 10, yPosition);
    yPosition += 5;
    doc.text(`Email: ${user.email}`, 10, yPosition);
    yPosition += 5;
    doc.text(`Phone: ${user.phone}`, 10, yPosition);
    yPosition += 10;

    doc.setLineWidth(0.5);
    doc.line(10, yPosition, 200, yPosition);
    yPosition += 10;

    const tableColumn = ["Item Name", "Quantity", "Price without VAT (€)",
    "VAT Amount (€)", "VAT %", "Total Price (€)"];
    const tableRows = [];

    purchaseGroup.items.forEach(purchase => {
        const {
            priceWithoutVAT,
            vatAmount,
            vatPercentage
        } = calculateVAT(parseFloat(purchase.total_price), purchase.item.vat);
        const purchaseData = [
            purchase.item.name,
            purchase.quantity,
            priceWithoutVAT,
            vatAmount,
            vatPercentage,
            purchase.total_price
        ];
        tableRows.push(purchaseData);
    });

    doc.autoTable({
        startY: yPosition,
        head: [tableColumn],
        body: tableRows,
        theme: 'grid',
        headStyles: {
            fillColor: [220, 220, 220],
            textColor: [0, 0, 0],
            fontStyle: 'bold'
        }
    });

    yPosition = doc.lastAutoTable.finalY + 10;

```

```

const totalPrice = purchaseGroup.totalPrice;
const {
  priceWithoutVAT: totalWithoutVAT,
  vatAmount: totalVAT
} = calculateVAT(totalPrice, purchaseGroup.items[0].item.vat);

const totalTableColumn = ["Description", "Amount (€)"];
const totalTableRows = [
  ["Total Price without VAT", totalWithoutVAT],
  ["Total VAT Amount", totalVAT],
  ["Total Price", totalPrice.toFixed(2)]
];

doc.autoTable({
  startY: yPosition,
  head: [totalTableColumn],
  body: totalTableRows,
  theme: 'grid',
  headStyles: {
    fillColor: [220, 220, 220],
    textColor: [0, 0, 0],
    fontStyle: 'bold'
  }
});

yPosition = doc.lastAutoTable.finalY + 20;

doc.setFontSize(12);
doc.text('Payment Terms and Conditions', 10, yPosition);
yPosition += 5;
doc.setFontSize(10);
doc.text('Payment is due within 30 days from the purchase date. After 30
days purchases will be canceled!', 10, yPosition);
yPosition += 10;

doc.setFontSize(12);
doc.text('Issuer:', 10, yPosition);
doc.text('Receiver:', 120, yPosition);
yPosition += 10;
doc.text('Name Surname: _____', 10, yPosition);
doc.text('Name Surname: _____', 120, yPosition);
yPosition += 5;
doc.text('Date: _____', 10, yPosition);
doc.text('Date: _____', 120, yPosition);
yPosition += 5;
doc.text('Signature: _____', 10, yPosition);
doc.text('Signature: _____', 120, yPosition);

doc.save(`purchase_history_${time}.pdf`);

```

```
};
```

RouterIndex.js

```
import { createRouter, createWebHistory } from 'vue-router';

const routes = [
  {
    path: '/',
    name: 'home',
    component: () => import('../views/HomeView/HomeView.vue'),
    meta: { tabName: 'Home' },
  },
  {
    path: '/login',
    name: 'login',
    component: () => import('../views/Auth/Login/Login.vue'),
    meta: { requiresGuest: true },
  },
  {
    path: '/register',
    name: 'register',
    component: () => import('../views/Auth/Register/Register.vue'),
    meta: { requiresGuest: true },
  },
  {
    path: '/admin',
    name: 'admin',
    component: () => import('../views/Admin/Dashboard/Admin.vue'),
    meta: { requiresAdmin: true },
  },
  {
    path: '/profile',
    name: 'profile',
    component: () => import('../views/Auth/Profile/Profile.vue'),
    meta: { requiresAuth: true },
  },
  {
    path: '/profile/edit',
    name: 'profile/edit',
    component: () => import('../views/Auth/EditData/EditData.vue'),
    meta: { requiresAuth: true },
  },
  {
    path: '/profile/change-password',
    name: 'profile/change-password',
    component: () =>
import('../views/Auth/ChangePassword/ChangePassword.vue'),
    meta: { requiresAuth: true },
  },
];
```

```

{
  path: '/admin/brands',
  name: 'admin/brands',
  component: () => import('../views/Admin/Brands/Brands.vue'),
  meta: { requiresAdmin: true }
},
{
  path: '/admin/brands/create',
  name: 'admin/brands/create',
  component: () => import('../views/Admin/Brands/Create.vue'),
  meta: { requiresAdmin: true }
},
{
  path: '/admin/brands/:id/edit',
  name: 'admin/brands/edit',
  component: () => import('../views/Admin/Brands/Edit.vue'),
  meta: { requiresAdmin: true }
},
{
  path: '/brands',
  name: 'brands',
  component: () => import('../views/Brands/Brands.vue'),
},
{
  path: '/brand/:id/products',
  name: 'brand-products',
  component: () => import('../views/BrandProducts/BrandProducts.vue'),
},
{
  path: '/products',
  name: 'products',
  component: () => import('../views/Products/Products.vue'),
},
{
  path: '/product/:id',
  name: 'product',
  component: () => import('../views/ProductView/ProductView.vue'),
},
{
  path: '/admin/items',
  name: 'admin/items',
  component: () => import('../views/Admin/Items/Items.vue'),
  meta: { requiresAdmin: true }
},
{
  path: '/admin/items/create',
  name: 'admin/items/create',
  component: () => import('../views/Admin/Items/Create.vue'),
  meta: { requiresAdmin: true }
},

```

```

{
  path: '/admin/items/:id/edit',
  name: 'admin/items/edit',
  component: () => import('../views/Admin/Items/Edit.vue'),
  meta: { requiresAdmin: true }
},
{
  path: '/admin/categories',
  name: 'admin/categories',
  component: () => import('../views/Admin/categories/categories.vue'),
  meta: { requiresAdmin: true }
},
{
  path: '/admin/categories/create',
  name: 'admin/categories/create',
  component: () => import('../views/Admin/categories/create-
categories.vue'),
  meta: { requiresAdmin: true }
},
{
  path: '/admin/categories/:id/edit',
  name: 'admin/categories/edit',
  component: () =>
import('../views/Admin/categories/EditCategories.vue'),
  meta: { requiresAdmin: true }
},
{
  path: '/admin/purchases',
  name: 'admin/purchases',
  component: () => import('../views/Admin/Purchases/Purchases.vue'),
  meta: { requiresAdmin: true }
},
{
  path: '/favorites',
  name: 'favorites',
  component: () => import('../views/Favorites/Favorites.vue'),
  meta: { requiresAuth: true }
},
{
  path: '/cart',
  name: 'cart',
  component: () => import('../views/Cart/Cart.vue'),
  meta: { requiresAuth: true }
},
{
  path: '/thank-you',
  name: 'thank-you',
  component: () => import('../views/ThankYouPage/ThankYouPage.vue'),
  meta: { requiresAuth: true }
},

```

```

    {
      path: '/admin/inventory',
      name: '/admin/inventory',
      component: () => import('../views/admin/Inventory/Inventory.vue'),
      meta: { requiresAdmin: true }
    },
    {
      path: '/admin/inventory-edit/:id/edit',
      name: '/admin/inventory-edit',
      component: () => import('../views/admin/Inventory/InventoryEdit.vue'),
      meta: { requiresAdmin: true }
    },
    {
      path: '/terms-of-service',
      name: '/terms-of-service',
      component: () => import('../views/Tos/Tos.vue'),
      meta: { requiresAuth: true }
    },
    {
      path: '/admin/specification-titles/:id/edit',
      name: '/admin/specification-titles/:id/edit',
      component: () =>
import('../views/Admin/categories/EditSpecifiationType.vue'),
      meta: { requiresAdmin: true }
    },
    {
      path: '/admin/specification-titles/add',
      name: '/admin/specification-titles/add',
      component: () =>
import('../views/Admin/categories/AddSpecificationType.vue'),
      meta: { requiresAdmin: true }
    },
  ],
];

const router = createRouter({
  history: createWebHistory(import.meta.env.BASE_URL),
  routes,
});

router.beforeEach((to, from, next) => {
  const isLoggedIn = !!localStorage.getItem('access_token');
  const user = JSON.parse(localStorage.getItem('user'));

  if (to.matched.some(record => record.meta.requiresAuth) && !isLoggedIn) {
    next({ name: 'login' });
  } else if (to.matched.some(record => record.meta.requiresAdmin) && (!user
|| user.admin !== 1)) {
    next({ name: 'login' });
  } else if (to.matched.some(record => record.meta.requiresGuest) &&
isLoggedIn) {

```



```

        next({ name: 'home' });
    } else {
        next();
    }
});

```

```
export default router;
```

Model Brands.php

```

<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Brands extends Model
{
    use HasFactory;

    protected $table = 'brands';

    protected $fillable = [
        'name',
        'img',
    ];

    public function items()
    {
        return $this->hasMany(Items::class, 'brand_id');
    }
}

```

Model Cart.php

```

<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Cart extends Model
{
    use HasFactory;

    protected $table = 'cart';
    protected $primaryKey = 'cart_id';

    protected $fillable = [

```

```

        'user_id',
        'item_id',
        'created_at',
        'updated_at'
    ];

    public function item()
    {
        return $this->belongsTo(Items::class, 'item_id');
    }
}

```

Model Categories.php

```

<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Categories extends Model
{
    use HasFactory;

    protected $table = 'categories';

    protected $fillable = [
        'category_name',
    ];

    public function items()
    {
        return $this->hasMany(Items::class, 'categories_id');
    }

    public function specificationTitles()
    {
        return $this->hasMany(SpecificationTitle::class, 'category_id');
    }
}

```

Model FavoriteItems.php

```

<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

```

```

class FavoriteItems extends Model
{
    use HasFactory;

    protected $table = 'favorite_items';
    protected $primaryKey = 'favorite_id';

    protected $fillable = [
        'user_id',
        'item_id',
        'created_at',
        'updated_at'
    ];

    public function item()
    {
        return $this->belongsTo(Items::class);
    }
}

```

Model Items.php

```

<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Items extends Model
{
    use HasFactory;

    protected $table = 'items';

    protected $fillable = [
        'name',
        'description',
        'price',
        'vat',
        'img',
        'brand_id',
        'categories_id',
        'amount',
        'reserved',
        'sold',
    ];

    protected $attributes = [

```

```

        'amount' => 0,
        'reserved' => 0,
        'sold' => 0,
    ];

    public function brands()
    {
        return $this->belongsTo(Brands::class);
    }

    public function category()
    {
        return $this->belongsTo(Categories::class, 'categories_id');
    }

    public function specificationDescriptions()
    {
        return $this->hasMany(SpecificationDescription::class, 'item_id');
    }
}

```

Model Purchase.php

```

<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Purchase extends Model
{
    use HasFactory;

    protected $fillable = [
        'user_id',
        'item_id',
        'total_price',
        'status',
        'quantity',
        'vat',
    ];

    public function item()
    {
        return $this->belongsTo(Items::class);
    }

    public function user()
    {
        return $this->belongsTo(User::class);
    }
}

```

```
}  
}
```

Model SpecificationDescription.php

```
<?php  
  
namespace App\Models;  
  
use Illuminate\Database\Eloquent\Factories\HasFactory;  
use Illuminate\Database\Eloquent\Model;  
  
class SpecificationDescription extends Model  
{  
    use HasFactory;  
  
    protected $table = 'specification_description';  
  
    protected $fillable = [  
        'description',  
        'item_id',  
        'specification_title_id',  
    ];  
  
    public function item()  
    {  
        return $this->belongsTo(Items::class, 'item_id');  
    }  
  
    public function specificationTitle()  
    {  
        return $this->belongsTo(SpecificationTitle::class,  
'specification_title_id');  
    }  
}
```

Model SpecificationTitle.php

```
<?php  
  
namespace App\Models;  
  
use Illuminate\Database\Eloquent\Factories\HasFactory;  
use Illuminate\Database\Eloquent\Model;  
  
class SpecificationTitle extends Model  
{  
    use HasFactory;  
  
    protected $table = 'specification_titles';
```

```

protected $fillable = [
    'specification_title',
    'category_id',
];

public function category()
{
    return $this->belongsTo(Categories::class, 'category_id');
}

public function specificationDescriptions()
{
    return $this->hasMany(SpecificationDescription::class,
'specification_title_id');
}
}

```

Model User.php

```

<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Foundation\Auth\User as Authenticatable;
use Illuminate\Notifications\Notifiable;
use Laravel\Sanctum\HasApiTokens;

class User extends Authenticatable
{
    use HasApiTokens, HasFactory, Notifiable;

    /**
     * The attributes that are mass assignable.
     *
     * @var array<int, string>
     */
    protected $fillable = [
        'username',
        'password',
        'name',
        'surname',
        'phone',
        'email',
        'admin',
    ];

    /**
     * The attributes that should be hidden for serialization.
     *
     * @var array<int, string>
     */
}

```

```

    */
protected $hidden = [
    'password',
    'remember_token',
];

/**
 * The attributes that should be cast.
 *
 * @var array<string, string>
 */
protected $casts = [
    'email_verified_at' => 'datetime',
];
}

```