**„RĪGAS VALSTS TEHNIKUMS”**

DATORIKAS NODAĻA

Izglītības programma: Programmēšana

**NOSLĒGUMA DARBS**

**Autoservisa sistēma**

Audzēknis: DP3-1 Jānis Gailis

SATURS

[SATURS 2](#_Toc165632682)

[IEVADS 3](#_Toc165632683)

[1. UZDEVUMA NOSTĀDNE 4](#_Toc165632684)

[1.1. SISTĒMAS LIETOTĀJI 5](#_Toc165632685)

[1.2. Uzdevuma risināšanas līdzekļu izvēles pamatojums 10](#_Toc165632686)

[2. DATU MODELĒŠANA 11](#_Toc165632687)

[2.1. Objektu orientēts konceptuālais datu modelis 11](#_Toc165632688)

[2.2. Entītiju relāciju datu modelis 14](#_Toc165632689)

[3. DATU BĀZES IZSTRĀDE 16](#_Toc165632690)

[3.1. Datu glabāšanas fiziskā struktūra 16](#_Toc165632691)

[3.2. Datu bāzes atbilstības pamatojums 3NF prasībām 18](#_Toc165632692)

[4. SISTĒMAS FUNKCIONĀLO PRASĪBU REALIZĒŠANA 19](#_Toc165632693)

[4.1. Datu pievienošana, labošana, dzēšana 19](#_Toc165632694)

[4.2. Datu validācija 26](#_Toc165632695)

[4.3. Datu izgūšana un apstrāde 28](#_Toc165632696)

[4.4. Datu integritātes nodrošināšanas metožu izmantošana 35](#_Toc165632697)

[4.4.1. Lietotāju autentifikācija 35](#_Toc165632698)

[4.4.2. Funkcionālo pienākumu sadalīsana 36](#_Toc165632699)

[NOBEIGUMS 37](#_Toc165632700)

[INFORMĀCIJAS AVOTI 38](#_Toc165632701)

[PIELIKUMS 39](#_Toc165632702)

[Datu bāzes kods: 39](#_Toc165632703)

IEVADS

Mūsdienās gandrīz katram ir sava automašīna un nepieciešams remonts, bet Latvijā ir maz pieejamas mājaslapas autoservisiem, kuras tiek aktīvi atjauninātas. Eksistējošajām mājas lapām ir slikts dizains un maz funkciju. Mana ideja ir veidot sistēmu, kura piedāvā ikdienas cilvēkiem ērtu, godīgu un ātru mašīnremontu.

Galvenie konkurenti ir sistēmas, kuras ļauj klientiem apskatīt autoservisu informāciju un sniedz klientiem kontaktinformāciju. Stiprās puses šīm sistēmām ir vienkāršs dizains, maz nepieciešamu funkciju, nav vajadzīga klientu profilu izveide, kā arī tās neizmanto daudz resursus, jo tikai uzrāda informāciju. Vājās puses konkurentiem, kuras mans projekts atrisinās, ir tas, ka liela daļa no konkurentiem izmanto novecojušus dizainus vai tādus, kurus klientiem grūti izprast. Mans projekts piedāvās arī papildus funkcijas, ar klientu reģistrācijas palīdzību spēsim piedāvāt remontu kalendāru, kas padarīs zvanīšanu nevajadzīgu, ietaupot laiku visiem. Citas funkcijas, kuras šo projektu atšķirs no konkurentiem ir cenrādis, kurš parāda aptuvenas cenas klientam uzreiz, jo klienti labi vērtē atklātību.

Šīs sistēmas aktualitāte tikai aug, jo arvien vairāk cilvēki izmanto mašīnas transportam un tām bieži nepieciešams remonts. Tieši mana sistēma ir nepieciešama, jo ar tās palīdzību iespējams ietaupīt laiku gan klientiem, gan mehāniķiem.

1. UZDEVUMA NOSTĀDNE

Noslēguma darba uzdevums ir autoservisa datu bāzes sistēmas izveide, kurā ietilpst lietotāju reģistrācijas sistēma, lai piešķirtu unikālas funkcijas tieši reģistrētiem lietotājiem un laika tabulas sistēmas izveide, kurā klienti var izvēlēties kalendārā redzamo brīvo laiku, lai pieteiktu savas mašīnas remontu.

Sistēmu nepieciešams veidot, lai ietaupītu laiku, gan darbiniekiem, gan klientiem un veidotu atklātību ar klientiem, arī palielinātu klientu skaitu autoservisam.

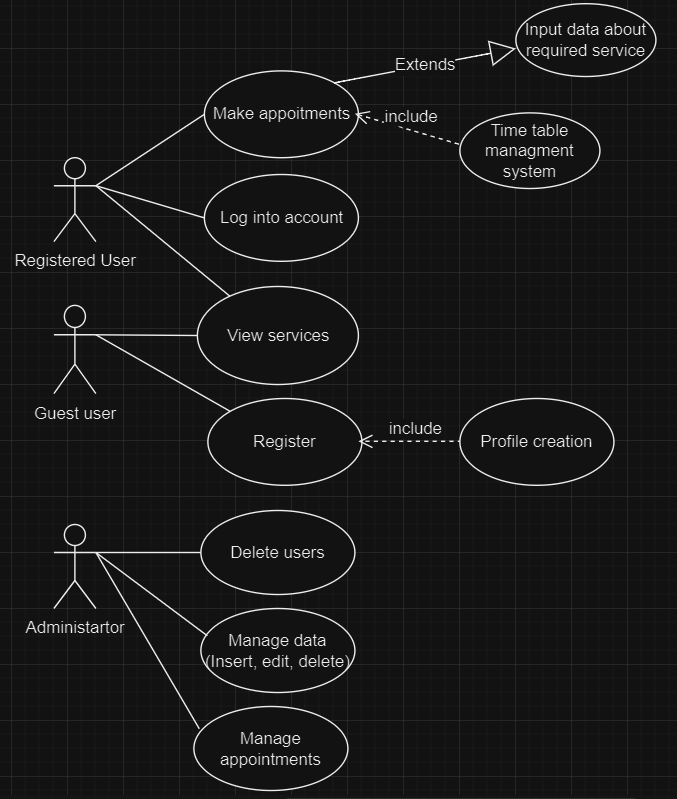
Autoservisa sistēmai jāveic šādas funkcijas.

* Atļaut lietotājiem izveidot un ieiet jau izveidotos profilos;
* Reģistrētiem lietotājiem piedāvāt papildus funkcijas, kuras citiem nav pieejamas (cenas kalkulators un kalendārs)
* Iespēju apskatīt aktīvos remontus;
* Pievienot datus kalendāra brīvajās vietās, lai pieteiktu remontu;
* Administratoram iespēja rediģēt kalendāra datus.
  1. SISTĒMAS LIETOTĀJI

Sistēma iedalīta 3 līmeņos.

* “Guest user”, jeb viesis, kurš var apskatīt piedāvājumus un reģistrēties.
* “Registered User”, jeb reģistrēts lietotājs, kurš papildus var arī veikt pieteikt remontu.
* “Administrator”, jeb pārvaldnieks, kurš var dzēst lietotājus un pārvaldīt informāciju par pakalpojumiem un pieteiktiem remontiem.

1.Shēma “Sistēmas lietotāji”



**Lietošanas gadījums “Guest User registers”**

**Pamatscenārijs.**

1. Lietotājs ievada e-pastu.

2. Lietotājs ievada vārdu.

3. Sistēma pārbauda, vai ievadītais e-pasts jau eksistē sistēmā, ja eksistē, tiek izsaukts izņēmums “Email already registered”.

4. Lietotājs ievada lietotājvārdu.

5. Sistēma pārbauda, vai ievadītais lietotājvārds jau eksistē sistēmā, ja eksistē, tiek izsaukts izņēmums “Username is taken”.

6. Lietotājs ievada paroli, sistēma pārbauda, vai tā ir pietiekami gara (at least 5 characters) un pietiekami droša (at least 1 uppercase and one special character)

7. Lietotājs atkārtoti ievada šo paroli.

8. Sistēma paziņo, ka sekmīgi reģistrēts lietotājs sistēmā.

**Izņēmuma scenārijs.**

“Email already registered”:

Sistēma atver aktivitāti ”atjaunināt lietotāja paroli”.

“Username is taken”:

Sistēma paziņo lietotājam, lai izvēlas citu lietotājvārdu.

**Lietošanas gadījums “Guest user logs in”**

**Pamatscenārijs**

1. Lietotājs ievada e-pastu vai lietotājvārdu.

2. Sistēma pārbauda vai e-pasts vai lietotājvārds sakrīt ar sistēmā esošajiem, ja nesakrīt, tiek izsaukts izņēmums “Credentials dont match any accounts”.

3.Lietotājs ievada paroli.

4. Sistēma pārbauda vai parole sakrīt ar paroli sistēmā, šim lietotājam, ja nesakrīt, tiek izsaukts izņēmums “Password is incorrect”.

5. Sistēma paziņo, ka lietotājs sekmīgi ienācis profilā.

**Izņēmuma scenārijs**

“Credentials dont match any accounts”:

Sistēma paziņo lietotājam, ka e-pasts vai lietotājvārds nesakrīt ar nevienu lietotāju sistēmā.

“Password is incorrect”;

Sistēma atver aktivitāti “atjaunināt lietotāja paroli”.

**Lietošanas gadījums “Logged in user makes an appoitment”**

**Pamatscenārijs**

1. Lietotājs izvēlas kāda veida pakalpojums nepieciešams (Quick repair, oil change, meet for organising big repairs or medium length services, like painting the car)

2. Sistēma atkarībā no izvēles maina laikus, kurus lietotājs var izvēlēties priekš remonta.(if its a medium length service, removes the options where its less than 2 days free)

3. Lietotājs īsi apraksta kas nepieciešams un izvēlas laiku kalendārā, ja izvēlētais laiks ir jau aizņemts, vai nav pietiekami daudz laiks priekš pakalpojuma, tad tiek izsaukts izņēmums “Invalid choice”

4. Sistēma šo laiku bloķē, lai cits lietotājs neizvēlas to pašu.

5. Sistēma paziņo gan klientam, gan darbiniekam par veikto pasūtījumu.

**Izņēmuma scenārijs**

“Invalid choice”:

Sistēma paziņo lietotājam, ka veiktā izvēle nav pieejama.

**Lietošanas gadījums “An appoitment was made, but the administrator notices that he needs to (add time, deny, or otherwise alter the information about it)”**

**Pamatscenārijs**

1. Administrators atver kalendāru un izvēlas noteiktu pakalpojuma laiku.

2. Sistēma parāda informāciju par to.

3. Administrators maina tā beigu datumu, papildinot to par 2 dienām.

4. Sistēma pārbauda vai tās 2 dienas ir brīvas, ja nav, tad tiek izsaukts izņēmums “Conflicting repair times”

5. Sistēma maina informāciju par pakalpojumu un informē klientu.

**Izņēmuma scenārijs**

“Conflicting repair times”:

Sistēma paziņo administratoram, ka laiks kurā grib ievietot šo darbu ir aizņemts.

**Lietošanas gadījums “Administrator changes repair costs”**

**Pamatscenārijs**

1. Administrators atver cenu sadaļu.

2. Izvēlas pogu, rediģēt un izmaina cenu.

3. Sistēma pārprasa, vai cenas maiņa ir pareizi.

4. Cena ir sistēmā mainīta

**Lietošanas gadījums “Client wants to check repair costs”**

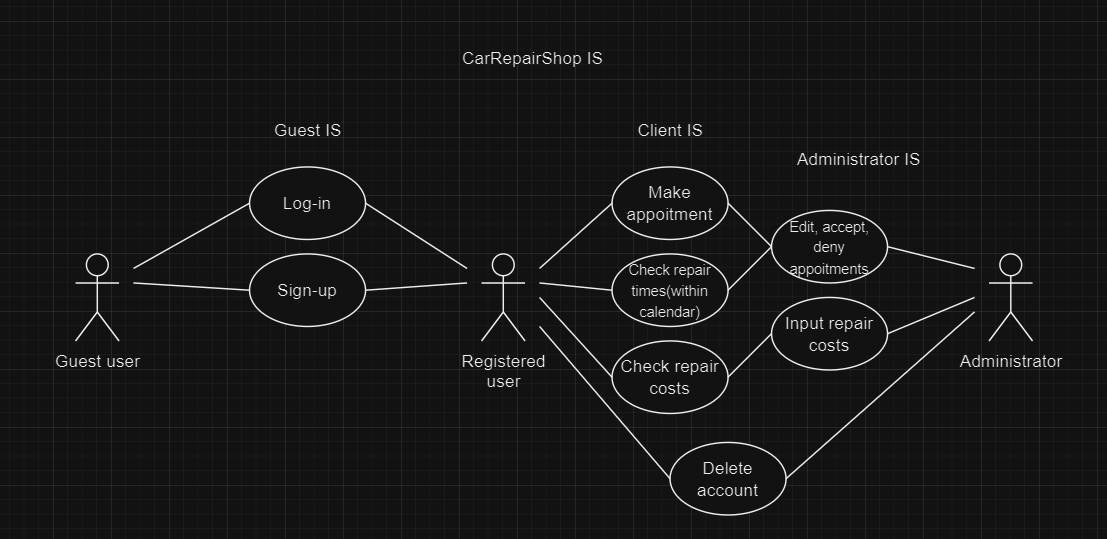
**Pamatscenārijs**

1. Lietotājs atver aktuālo sadaļu.

2. Atzīmē kuri no pakalpojumiem nepieciešami (ja tā ir krāsošana izvēlas kuras mašīnas daļas nepieciešams krāsot)

3. Sistēma saskaita visus kopā un izvada kopsummu.

2.Shēma “Sistēmas lietotāju scenāriji”



* 1. Uzdevuma risināšanas līdzekļu izvēles pamatojums

Sistēma ir paredzēta pārlūkprogrammai, tādēļ to realizācijai tiek izmantoti jaunākie programmatūras risinājumi.

Datu bāzes izstrādei un administrācijas panelim tika izmantota MySQL Workbench 8.0.36 CE, 2024. gada versija, jo MySQL Workbench ir plaši izmantots vizuālās datu bāzes dizaina rīks un administrēšanas rīks. Tas piedāvā lietotājam draudzīgu saskarni MySQL datu bāzu projektēšanai, pārvaldībai un mijiedarbībai ar tām.

Datu bāzes savienošanai ar tīmekļa vietni tiek izmantots Node.js (v20.11.0), galvenokārt izmantots, jo dod iespēju izmantot JavaScript gan klienta, gan servera pusē, kas atvieglo koda koplietošanu un paātrina izstrādi.

Serverim tika izmantots Express.js (v.4.18.2), jo tas ir tīmekļa lietojumprogrammu ietvars Node.js, kas nodrošina spēcīgu funkciju kopumu tīmekļa izveidei un tas ir vienkāršs un elastīgs. Express starpprogrammatūras sistēma ļauj viegli integrēt trešo pušu moduļus, ļaujot lietojumprogrammai pievienot dažādas funkcijas un tā nodrošina vienkāršu un efektīvu maršrutēšanas sistēmu HTTP pieprasījumu apstrādei.

1. DATU MODELĒŠANA
   1. Objektu orientēts konceptuālais datu modelis

Tiek izveidotas 5 klases:

“**User\_instance**” klase, kurā ir informācija par projektu kopumu un pašreizējo datumu.

Ar atribūtiem:

* “UUID”- kas unikālu ID, priekš lietotāja login gadījuma;
* “idUser”- kas saglabā lietotāja identifikācijas numuru;
* “InstaceStart” – kas saglabā gadījuma sākuma laiku;

Metodes:

* AddInstance(idUser:INT):VOID – Pievieno jaunu login gadījumu
* RemoveInstance(UUID:INT):VOID – Noņem gadījumu no tabulas

“**Projects**” klase, kurā ir informācija par visiem projektiem, kuri var būt dažādās stadijās.

Ar atribūtiem:

* “idProject” – kas saglabā projekta identifikācijas numuru;
* “idTime\_Table” – kas saglabā kalendāra identifikācijas numuru, kurā ir šis projekts;
* “idUser” – kas saglabā lietotāja identifikācijas numuru, kurš saistīts ar šo projektu;
* “StartDate” – kas saglabā informāciju par projekta sākuma datumu;
* “EndDateProjection” – kas saglabā informāciju par to, kad plānots pabeigt projektu;
* “Delayed” – kas saglabā to vai projekts tiks kavēts;
* “ProjectInfo” – kas saglabā projekta apraksta informāciju (kas jāizdara).

Metodes:

* ChangeEndDate(DATE):VOID – Nomaina projekta beigu datumu
* Delayed():VOID – Maina projekta statusu “delayed”, kas informē klientu par kavēšanos
* GetProjecInfo():VARCHAR – Izvada informāciju par projektu
* EditProjectInfo(VARCHAR):VOID – Rediģē informāciju par projektu.

“**Users**” klase, kurā ir informācija par visiem lietotājiem.

Ar atribūtiem:

* “idUser” – kas saglabā lietotāja identifikācijas numuru;
* “Name” – kas saglabā lietotāja vārdu;
* “Username” – kas saglabā lietotāja lietotājvārdu;
* “Email” – kas saglabā lietotāja e-pastu;
* “Password” – kas saglabā lietotāja paroli.

Metodes:

* ChangeUsername(VARCHAR):VOID – Nomaina lietotājvārdu;
* ChangePassword(VARCHAR):VOID – Nomaina paroli;
* GetUserInfo():VARCHAR – Izvada informāciju par lietotāju;
* DeleteUser():VOID – Dzēš lietotāju.

“**Workers**” klase, kurā ir informācija par visiem darbiniekiem.

Ar atribūtiem:

* “idUser” – kas saglabā lietotāja identifikāciju numuru, kurš saistīts ar šī darbinieka numuru (katram darbiniekam arī ir profils kurā var ienākt)
* “tenure” – kas saglabā darbinieka stāžu
* “StartWorkDate” – kas saglabā datumu, kad darbinieks sāka strādāt uzņēmumā
* “WorkType” – kas saglabā darbinieka lomu uzņēmumā

Metodes:

* RemoveTenure():VOID – Noņem stāžu no darbinieka (gadījumā ja tiek atlaists vai aiziet no darba)
* AddWorker(Email:VARCHAR):VOID – Pievieno darbinieku;
* RemoveWorker():VOID – Noņem darbinieku;

“**Administrators** “klase, kurā informācija par visiem darbiniekiem ar administratīvām tiesībām t.i. lietotāju dzēšana un projektu informācijas rediģēšana

Ar atribūtiem:

* “idUser”- kas saglabā lietotāja identifikācijas numuru, kas ir arī administratora id;
* “AdminTenure” – kas saglabā cik ilgi darbinieks ir bijis administrators;

Metodes:

* AddWorker(Email:VARCHAR):VOID – Pievieno darbinieku;
* RemoveWorker():VOID – Noņem darbinieku;
* RemoveTenure():VOID – Noņem stāžu no darbinieka (gadījumā ja tiek atlaists vai aiziet no darba)

Attiecības starp klasēm:

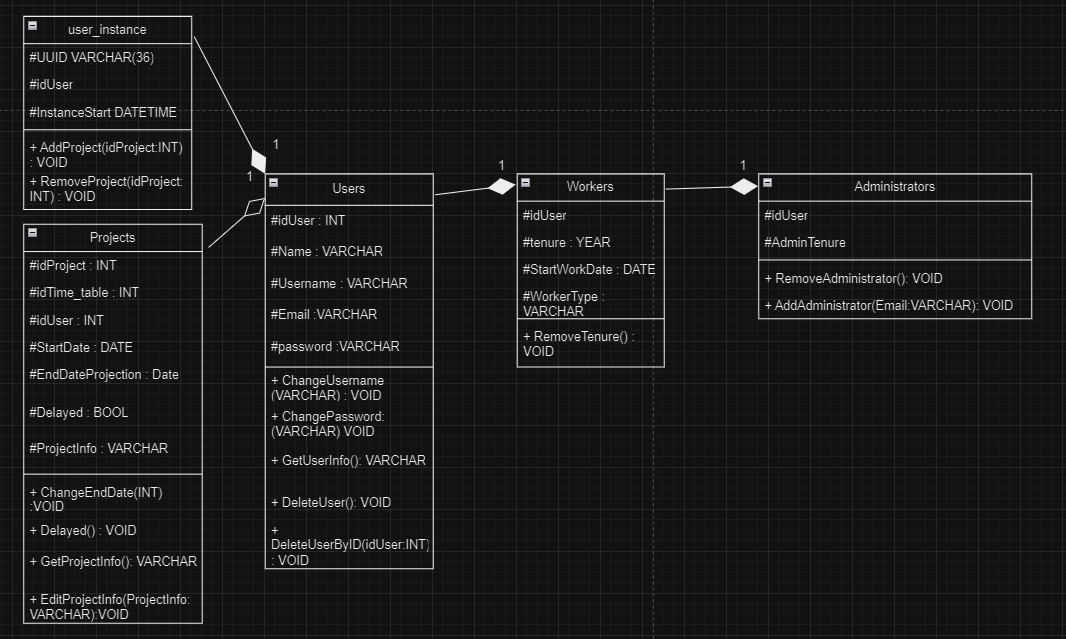
“Administrators” klase ir kompozīcija no klases “Worker”, jo katrs pārvaldnieks būs arī darbinieks, bez informācijas par šo darbinieku, viņš nevar būt pārvaldnieks;

“Workers” klase ir kompozīcija no klases “Users”, jo darbinieks nevar eksistēt bez profila, tas tikai vēlāk tiek saistīts ar informāciju par darbinieku;

“Users” klase ir agregācija no klases “Projects”, jo gan projekti gan lietotāji var eksistēt atsevišķi, bet lietotājs var izveidot projektu, kas būs saistīts tieši ar viņu, šis projekts tiks pabeigts, bet lietotājs paliks;

“user\_instance” klase ir ir kompozīcija no klases “Users”, jo vispirms ir lietotājs, kurš var ieiet profilā tādejādi sākot gadījumu.

3.Shēma “Datu modelis”



## Entītiju relāciju datu modelis

Datu bāzes projektēšana datu kopu un saišu starp tām attēlošanai tika lietota realitāšu-saišu diagramma, kas sastāv no divu veidu objektiem – entītija (loģiskais objekts) un relācijas (attiecības). ER modelis sastāv no 5 entītijām, kas atspoguļo datu apriti sistēmā.

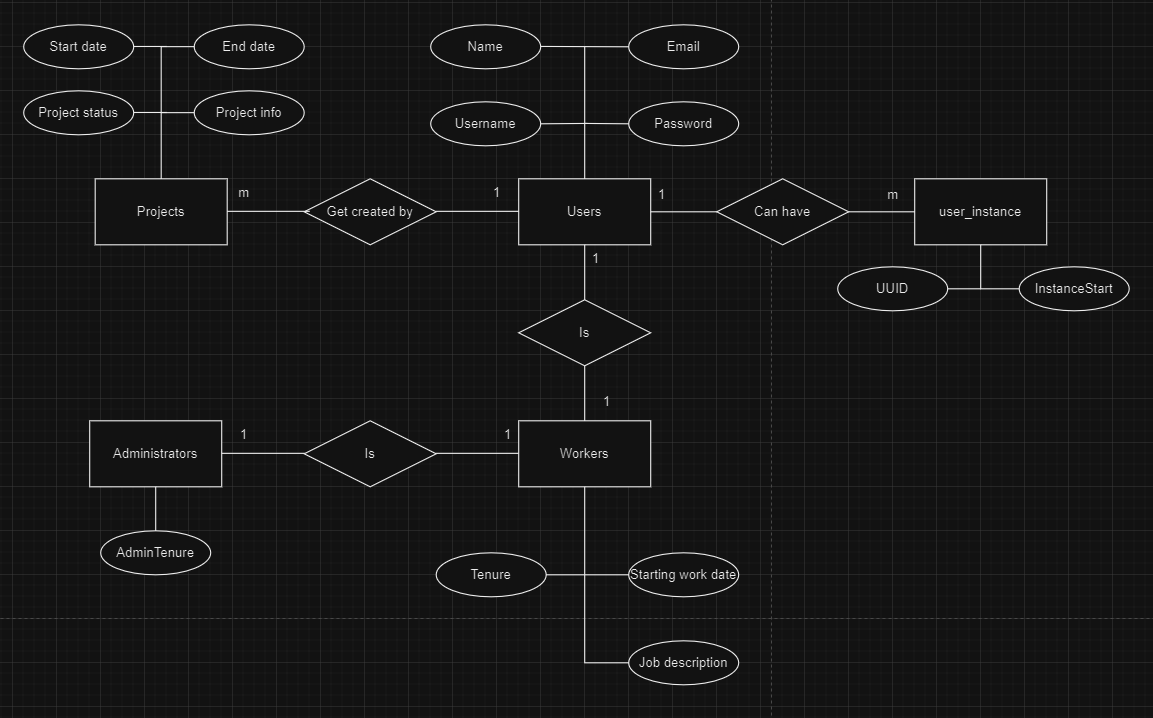
Lai ieviestu datu bāzes risinājumu, mēs izstrādājām sekojošo entītiju attiecību modeli:

* "User\_instance": Glabā informāciju par lietotāja ielogošanās gadījumu, lai sistēmā būtu zināms vai kāds lietotājs pašlaik izmanto sistēmu un spētu piešķirt atbilstošās funkcijas. Tās atribūtu kopums sevī ietver unikālu un sarežģītu ID un laiku kurā sākts šis gadījums.
* "Projects": Ietver informāciju par projektiem. Tās atribūtu kopums sevī ietver projekta sākuma, beigu datumus, projekta aprakstu un statusu.
* "Users": Glabā informāciju par lietotājiem. Tās atribūtu kopums sevī ietver lietotāja vārdu, lietotājvārdu, e-pastu un paroli.
* "Worker": Glabā informāciju par darbiniekiem. Tās atribūtu kopums sevī ietver darba stāžu, strādāšanas sākuma datumu un darba aprakstu.
* "Administrator": Glabā informāciju par administratoriem. Tās atribūtu kopums sevī ietver stāžu, cik ilgi ir bijis administrators.

Datu bāzes relācijas uzrāda kā savstarpēji savienotas divas vai vairākas entītijas.

* Starp lietotāja atgadījumu tabulas un lietotājiem ir attiecība viens pret vairākiem, jo vienam lietotājam var būt vairāki atgadījumi, bet viens atgadījums var būt tikai vienam lietotājam.
* Starp projektiem un lietotājiem ir attiecība viens pret vairākiem, jo viens pasūtījums var būt tikai vienam lietotājam, bet klientam var būt vairāki pasūtījumi.
* Starp lietotājiem un darbiniekiem ir attiecība viens pret viens, jo viens lietotājs sistēmā var skaitīties par vienu darbinieku un viens darbinieks sistēmā ir viens lietotājs.
* Starp administratoriem un darbiniekiem ir attiecība viens pret vienu, jo administrators sistēmā skaitās kā viens darbinieks un viens darbinieks var būt tikai viens administrators.

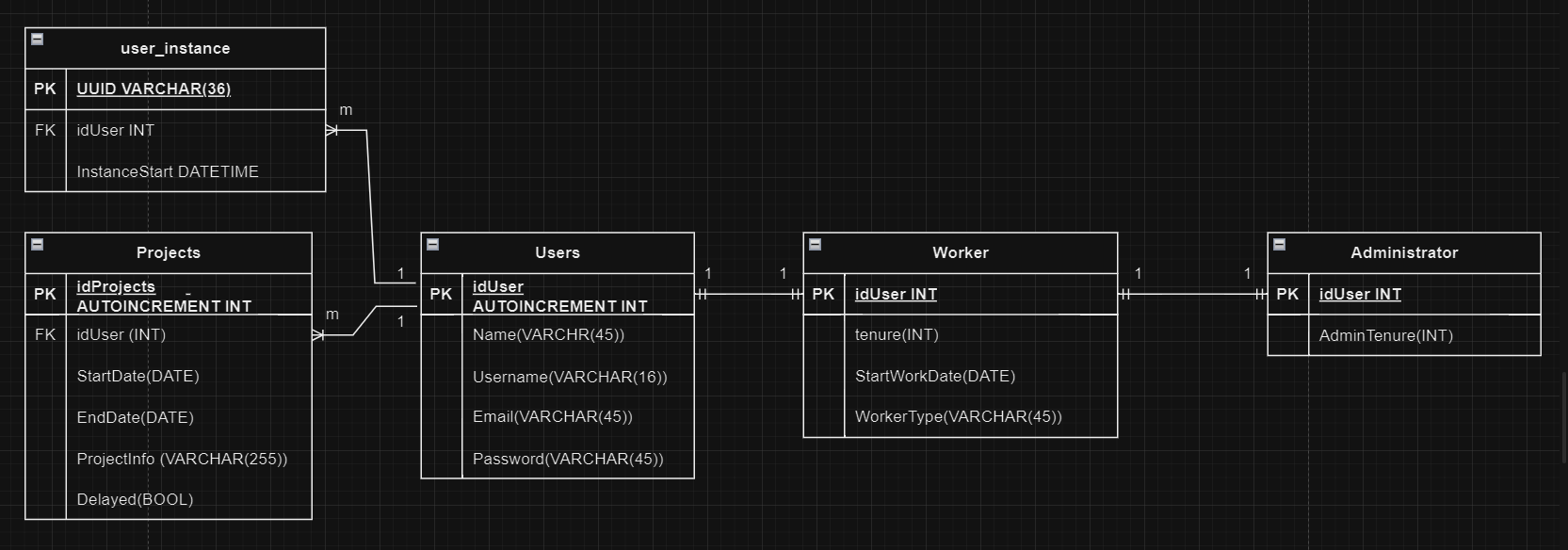
4.Shēma “ER diagramma”



# DATU BĀZES IZSTRĀDE

## Datu glabāšanas fiziskā struktūra

Datu bāze sastāv no 5 tabulām, kas satur informāciju par lietotājiem, lietotāju pieteikšanās gadījumiem , darbiniekiem, administratoriem un projektiem. Datubāzes projektēšanas procesā, lai tabulas atbilstu realizējamiem uzdevumiem un tiktu novērsta datu dublēšanās, tiek veikta to vienkāršošana un optimizēšana jeb normalizācija. Zemāk dotajā attēlā ir parādīta datu bāzes struktūra, kas demonstrē datu kolonu izvietojumu pa tabulām, kā arī tabulu savstarpējo saistību:

5.Shēma “Datu bāzes tabulu shēma”  


Tabula “user\_instance” glabā informāciju par sistēmā aktīvajiem lietotājiem: Unikālu identifikatora kodu izmantojot UUIDv4, kā arī lietotāja kodu ar kuru saistīts šis pieslēgšanās gadījums un laiks kurā iesākts gadījums.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Nr. | Nosaukums | Tips | Garums | Piezīme |
| 1 | UUID | VARCHAR | 36 | Unikāli ģenerēts kods, izmantots kā primārā atslēgā |
| 2 | idUser | INT |  | Ārējā atslēga, kas savieno ar attiecīgo lietotāju |
| 3 | InstanceStart | DATETIME |  | Datums un laiks kurā lietotājs uzsāka darbību pašreizējajā pieslēgšanās gadījumā |

1.Tabula “user\_instance struktūra”

Tabula “Users” glabā informāciju par reģistrētajiem lietotājiem: Unikālu identifikatora kodu, vārdu, lietotājvārdu, ēpasta adresi un paroli.

2.Tabula “Users struktūra”

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Nr. | Nosaukums | Tips | Garums | Piezīme |
| 1 | idUser | INT |  | Lietotāja kods, lietots kā primārā atslēga |
| 2 | Name | VARCHAR | 15 | Lietotāja legālais vārds |
| 3 | Username | VARCHAR | 20 | Lietotāja lietotājvārds, kas tiks izmantots, lai tiktu profilā |
| 4 | Email | VARCHAR | 45 | Lietotāja ēpasta adrese formāts [test@test.com](mailto:test@test.com) |
| 5 | Password | VARCHAR | 45 | Lietotāja parole |

Tabula “Worker” glabā informāciju par reģistrētajiem lietotājiem: Lietotāja unikālo identifikatora kodu, stāžu, Datumu kurā uzsākts strādāt un darbinieka veidu.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Nr. | Nosaukums | Tips | Garums | Piezīme |
| 1 | idUser | INT |  | Lietotāja kods, lietots kā primārā atslēga |
| 2 | tenure | INT |  | Darbinieka stāža |
| 3 | StartWorkDate | DATE |  | Datums kurā uzsākts strādāt uzņēmumā |
| 4 | WorkerType | VARCHAR | 45 | Darbinieka lomas nosaukums |

3.Tabula “Worker struktūra”

Tabula “Administrator” glabā informāciju par reģistrētajiem lietotājiem: Lietotāja unikālo identifikatora kodu un stāžu cik ilgi bijis administrators.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Nr. | Nosaukums | Tips | Garums | Piezīme |
| 1 | idUser | INT |  | Lietotāja kods, lietots kā primārā atslēga |
| 2 | AdminTenure | INT |  | Administratora stāža |

4.Tabula “Administrator struktūra”

Tabula “Projects” glabā informāciju par reģistrētajiem lietotājiem: Lietotāja unikālo identifikatora kodu un stāžu cik ilgi bijis administrators.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Nr. | Nosaukums | Tips | Garums | Piezīme |
| 1 | idProjects | INT |  | Lietotāja kods, lietots kā primārā atslēga |
| 2 | idUser | INT |  | Administratora stāža |
| 3 | StartDate | VARCHAR | 45 | Datums un laiks kurā sāksies projekts ISO laika formāts, lai vieglāk API ar to darboties |
| 4 | EndDateProjection | VARCHAR | 45 | Datums un laiks kurā plānots pabeigt projekti, ISO laika formāts, lai vieglāk API ar to darboties |
| 5 | Delayed | BOOL |  | Nepieciešams gadījumos, kad projekts aizņems vairāk laika nekā sākotnēji bija plānots. |
| 6 | ProjectInfo | VARCHAR | 255 | Papildus informācija ko lietotājs ievadīja par projektu |

4.Tabula “Projects struktūra”

## Datu bāzes atbilstības pamatojums 3NF prasībām

Atbilstība 1NF:

* Atribūti ir parasti mainīgie, nevis masīvi;
* Katram saistīto datu komplektam ir izveidota sava tabula;
* Katrs datu komplekts identificējams ar primāro atslēgu.

Atbilstība 2NF:

* Atbilst 1NF;
* Izveidotas atsevišķas tabulas tādām vērtību kopām, kas attiecas uz vairākiem ierakstiem;
* Sasaista tabulas, izmantojot ārējās atslēgas.

Atbilstība 3NF:

* Atbilst 2NF;
* Katrs lauks atkarīgs no atslēgas (neprimārie atribūti nav atkarīgi viens no otra).

# SISTĒMAS FUNKCIONĀLO PRASĪBU REALIZĒŠANA

## Datu pievienošana, labošana, dzēšana

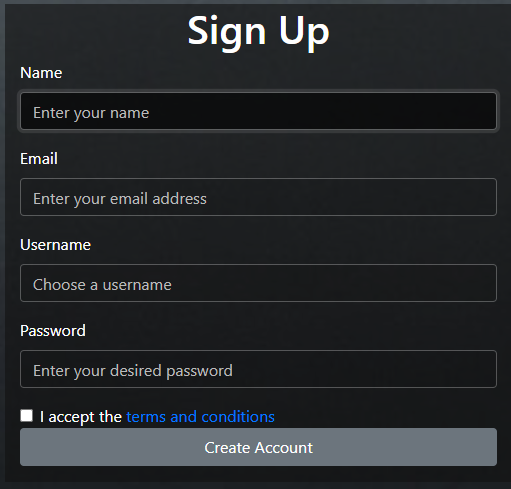
Datu bāze sastāv no 5 tabulām, kas satur informāciju par lietotājiem, lietotāju pieteikšanās gadījumiem, darbiniekiem, administratoriem un projektiem.

* Lietotāju tabula ir vienīgā ar kuras datiem spēj darboties parastais lietotājs, vispirms pievienot datus izveidojot kontu, tad rediģēt tos, mainot savu parole. Bet kontu dzēšana ir iespējama tikai administratoriem.
* Lietotāj pieteikšanās tabulā dati tiek pievienoti automātiski, kad lietotājs veic pieslēgšanos, kā arī tiek dzēsti, kad lietotājs atslēdzas. Bet papildus administrators spēj dzēst lietotāja kontus, lai to veiktu automātiski tiek atvienots arī lietotājs no pieslēgšanās tabulas.
* Ar darbinieku tabulas datiem var darboties tikai administrators, vispirms aizpildot formu ievada datus, pēc tam var tos rediģēt vai dzēst darbinieku.
* Administratoru tabulas datus arī rediģē tikai administratori, pievienojot darbinieku viņi izvēlas vai šis darbinieks ir arī administrators un tādejādi tiks aizpildīta administratora tabula.
* Projektu tabulas datus vispirms spēj ievadīt jebkurš lietotājs, kurš nav darbinieks, aizpildot attiecīgo formu. Pēc tam administrator var rediģēt vai dzēst šos projektus.

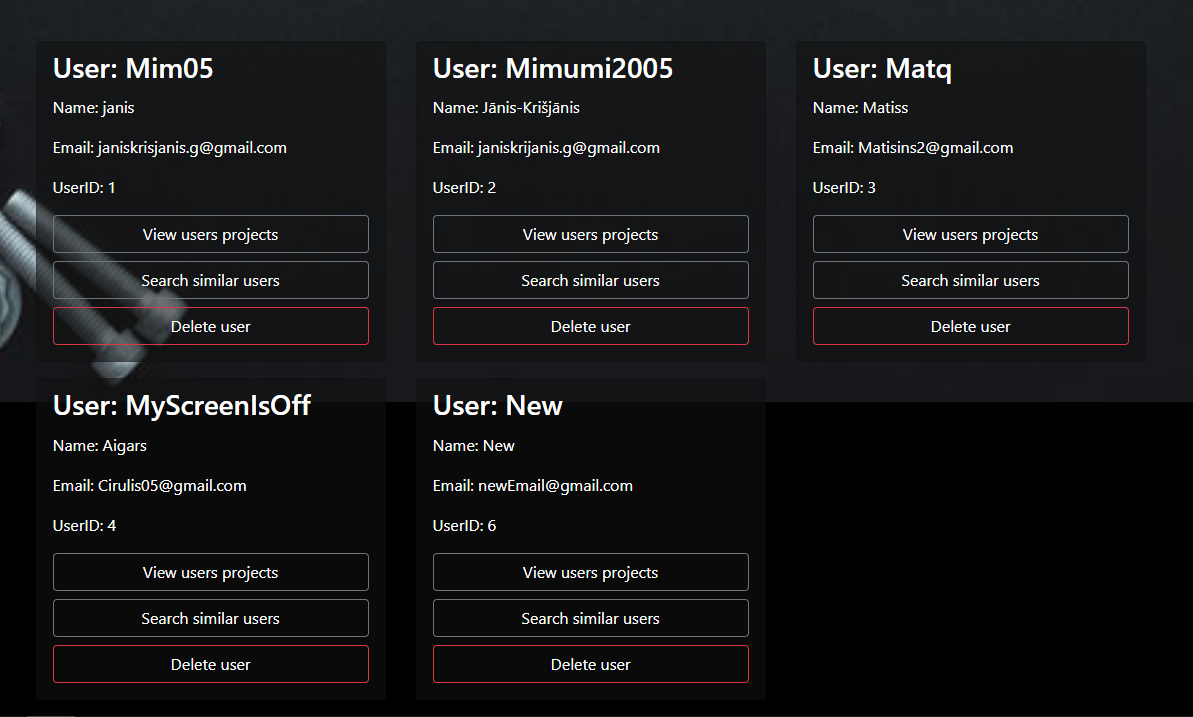
Zemāk sīkāk aprakstīti lietotāju tabulas datu pievienošanas, labošanas un dzēšanas gadījumi:

Lietotāju tabulā dati tiks ievadīti, kad lietotājs veido savu profilu un aizpilda attiecīgo formu:

1.att “Reģistrācijas form”



2.att “Dati pirms reģistrācijas”



Lai datu bāzē saglabātu informāciju vispirms tiek pārbaudīts vai informācija nesakrīt ar jau esošajiem ierakstiem, pēc tam, ja ieraksti nesakrīt, tiek reģistrēts jaunais lietotājs zemāk redzams kods:

1.kods “Reģistrācija”

app.post("/signup", (req, res) => {

  const { name, email, username, password } = req.body;

  // Check if the email is already taken

  const emailCheckQuery = 'SELECT \* FROM users WHERE email = ?';

  connection.query(emailCheckQuery, [email], (emailErr, emailResults) => {

    if (emailErr) {

      console.error('Error checking email in the database:', emailErr);

      return res.status(500).json({ status: 'error', message: 'Internal Server Error' });

    }

    if (emailResults.length > 0) {

      // Email is already taken

      return res.status(409).json({ status: 'error', message: 'Email is already taken' });

    }

    // Check if the username is already taken

    const usernameCheckQuery = 'SELECT \* FROM users WHERE username = ?';

    connection.query(usernameCheckQuery, [username], (usernameErr, usernameResults) => {

      if (usernameErr) {

        console.error('Error checking username in the database:', usernameErr);

        return res.status(500).json({ status: 'error', message: 'Internal Server Error' });

      }

      if (usernameResults.length > 0) {

        // Username is already taken

        return res.status(409).json({ status: 'error', message: 'Username is already taken' });

      }

      // Continue with the signup process

      const sql\_query = 'INSERT INTO users (name, email, username, password) VALUES (?, ?, ?, ?)';

      connection.query(sql\_query, [name, email, username, password], (err, result) => {

        if (err) {

          console.error('Error inserting data into the database:', err);

          return res.status(500).json({ status: 'error', message: 'Internal Server Error' });

        }

        loginUser(username, password, connection, res);

      });

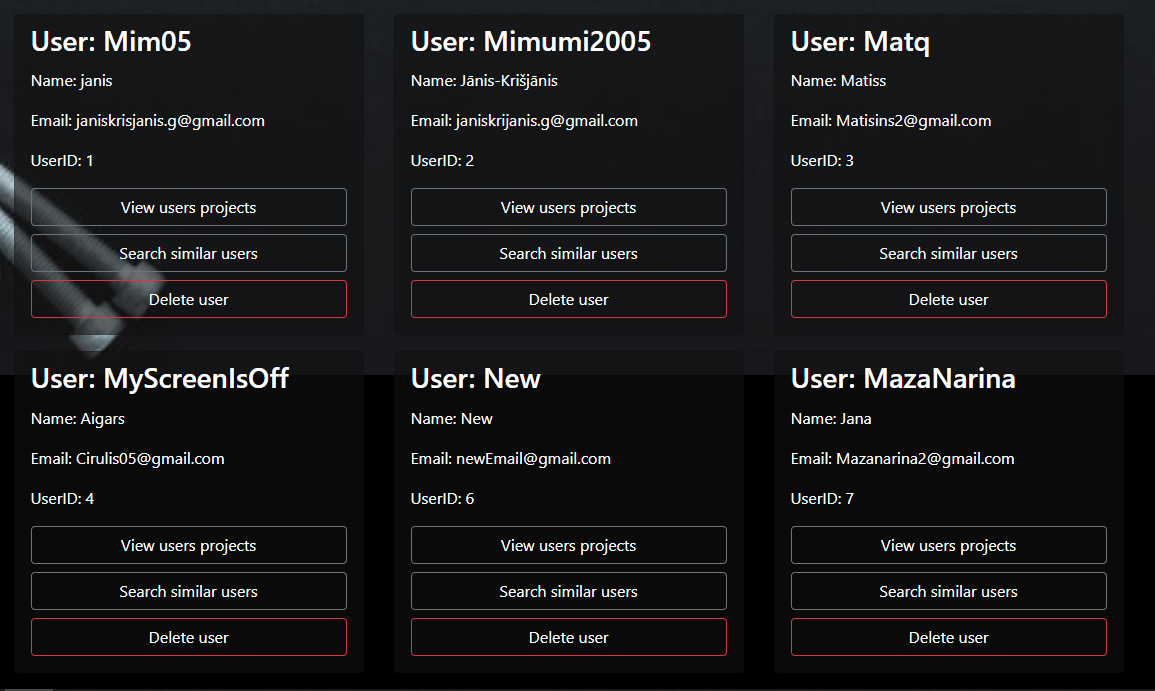
  });

});

});

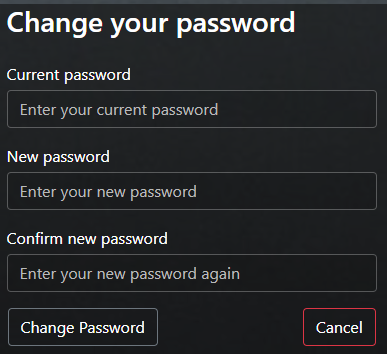
Kad šis kods tiek izpildīts, datu bāzē ir ievadīti jauni dati zemāk redzama informācija par jauno lietotāju:

3.att”Dati pēc reģistrācijas”



Lai rediģētu datus lietotājs aizpilda citu formu, kurā tiek mainīta parole, bet tā kā paroles informācija ir privātā informācija, tā netiek rādīta administratoriem, tāpēc nemainās informācija kas rādās par lietotājiem, pat ja tikusi mainīta parole.

4.att. “Paroles maiņa”



Mainot parole, vispirms tiek pārbaudīts vai lietotājs ID sakrīt ar to kas saistīta ar UUID kas saglabāts datu bāzē. Tad tiek pārbaudīts vai paroles sakrīt, ja sakrīt tiek mainīta reģistrētā parole uz jauno.

2.kods. “Paroles maiņas kods”

app.post("/change-password", (req, res) => {

  const { currentPassword, newPassword, UUID } = req.body;

  // Check user based on UUID

  const checkUserQuery = 'SELECT idUser FROM user\_instance WHERE idInstance = ?';

  connection.query(checkUserQuery, [UUID], (err, results) => {

    if (err) {

      console.error('Error checking user in the database:', err);

      return res.status(500).json({ status: 'error', message: 'Internal Server Error' });

    }

    // Check if UUID corresponds to a user

    if (results.length === 0) {

      return res.status(401).json({ status: 'error', message: 'Invalid UUID' });

    }

    // Retrieve the idUser from the results

    const loggedInUser = results[0].idUser;

    // Check if the current password matches the user's current password

    const checkCurrentPasswordQuery = 'SELECT \* FROM users WHERE idUser = ? AND password = ?';

    connection.query(checkCurrentPasswordQuery, [loggedInUser, currentPassword], (err, results) => {

      if (err) {

        console.error('Error checking current password in the database:', err);

        return res.status(500).json({ status: 'error', message: 'Internal Server Error' });

      }

      if (results.length === 0) {

        // Incorrect current password

        return res.status(401).json({ status: 'error', message: 'Incorrect current password' });

      }

      // Update the user's password with the new one

      const updatePasswordQuery = 'UPDATE users SET password = ? WHERE idUser = ?';

      connection.query(updatePasswordQuery, [newPassword, loggedInUser], (updateErr) => {

        if (updateErr) {

          console.error('Error updating password in the database:', updateErr);

          return res.status(500).json({ status: 'error', message: 'Internal Server Error' });

        }

        res.json({ status: 'success', message: 'Password updated successfully!' });

      });

    });

  });

});

Lai dzēstu lietotāju, administrators apskata lietotājus un viņam ir dota iespēja tos dzēst ja nepieciešams.

5.att. “Pirms lietotāja dzēšanas”



Lai dzēstu lietotāju vispirms tiek dzēsti visi ar šo lietotāju saistītie projekti, kā arī ja lietotājs ir pašlaik pieslēdzies, viņa pieslēgšanās gadījums tiek dzēsts un tad dzēš pašu lietotāju.

3.kods “Lietotāja dzēšanas kods”

// Route to delete a user and all associated projects

app.delete('/user-delete/:userID', async (req, res) => {

  const userID = req.params.userID;

  try {

    // Fetch the UUID associated with the user ID

    const uuidQuery = 'SELECT idInstance FROM user\_instance WHERE idUser = ?';

    connection.query(uuidQuery, [userID], async (err, results) => {

      if (err) {

        console.error('Error fetching UUID:', err);

        return res.status(500).json({ message: 'Error deleting user', error: err.message });

      }

      // Log out the user using the UUID if it exists

      if (results.length > 0) {

        await logoutUser(results[0].idInstance);

      }

      // Delete all projects associated with the user

      await deleteProjects(userID);

      // Proceed to delete the user from the users table

      const deleteUserQuery = 'DELETE FROM users WHERE idUser = ?';

      connection.query(deleteUserQuery, [userID], (error, deleteResult) => {

        if (error) {

          console.error('Error deleting user:', error);

          return res.status(500).json({ message: 'Error deleting user', error: error.message });

        }

        console.log('User deleted successfully');

        res.status(200).json({ message: 'User deleted successfully' });

      });

    });

  } catch (error) {

    console.error('Error deleting user:', error);

    return res.status(500).json({ message: 'Error deleting user', error: error.message });

  }

});

6.att “Pēc lietotāja dzēšanas”

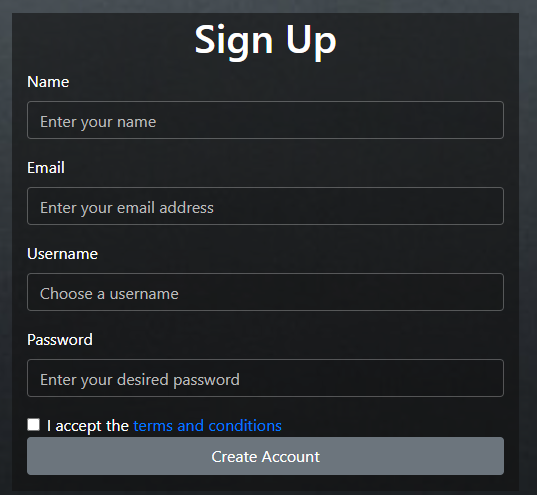


## Datu validācija

Katrā formā ko lietotājs aizpilda ir dažādu veidu datu validācija, kas rūpējās par to, lai lietotājs ievadītu pareizu informāciju, kura ir derīga datu bāzei.

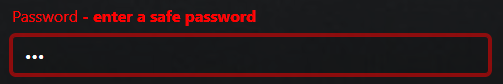
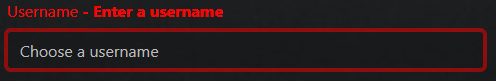
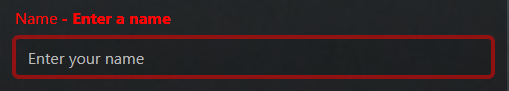
Veidojot jaunu kontu, lietotājam jāizpilda reģistrācijas forma, kas redzama zemāk:

15.att “Reģistrācijas forma”



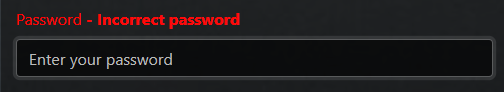
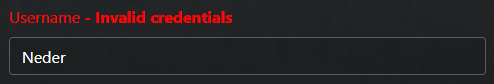
Katrā ievades daļā notiek vairākas pārbaudes, vai lietotājs ir ievadījis derīgu informāciju, zemāk redzams, kā tiek paziņots lietotājam, ja informācija, kas ievadīta nav derīga:

16.att”Nederīga reģistrācijas informācija”



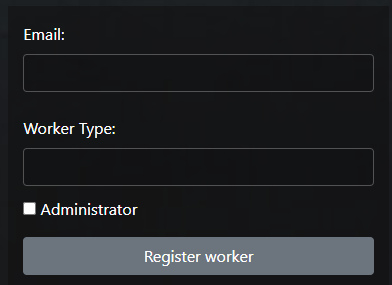
Līdzīgi darbojas arī pieslēgšanās profilam:

17.att “Nederīga pieslēgšanās informācija”



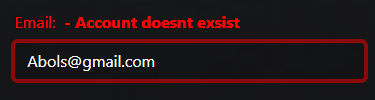
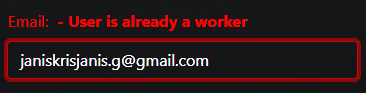
Arī administratora darbībām ir līdzīga autentifikācija, piemēram, zemāk redzams ekrānuzņēmums no darbinieku pievienošanas formas:

18.att “Darbinieku pievienošanas forma”



Tādā pašā veidā tiek uzrādīta kļūda, ja dati ir nederīgi:

19.att “Nederīga darbinieku pievienošanas informācija”

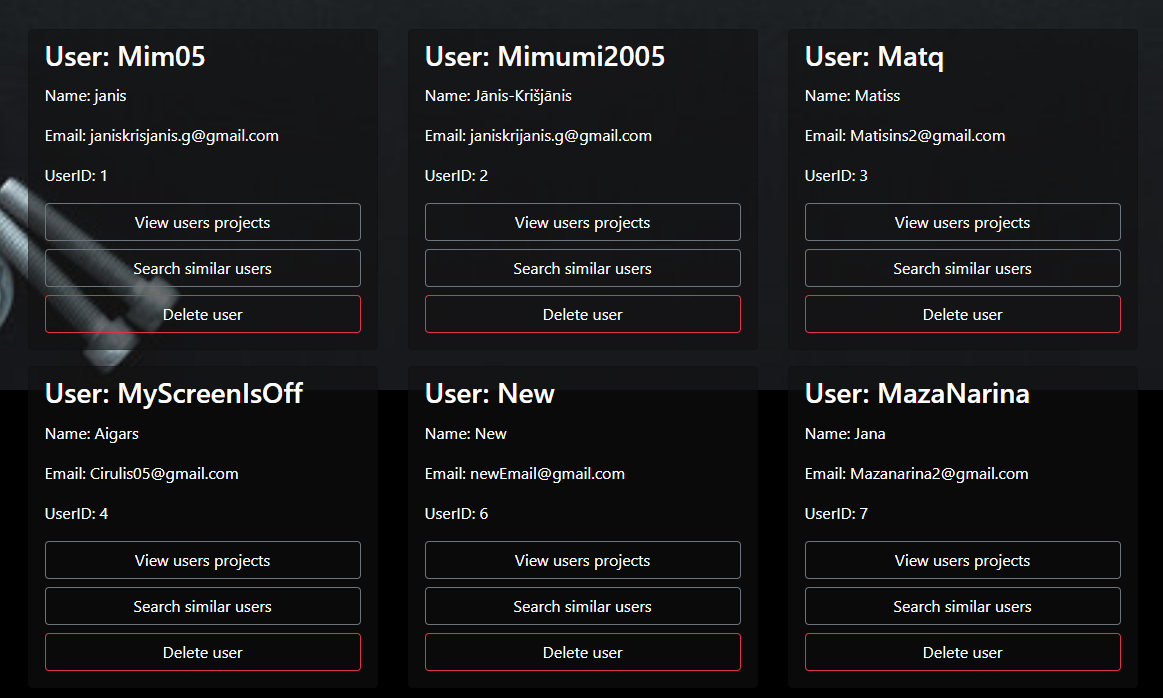


## Datu izgūšana un apstrāde

Lai administrators efektīvi spētu apskatīt datu bāzes datus, tiek izmantoti dažādi query, kas izvada nepieciešamo informāciju atdalot nevajadzīgo vai savienojot tabulas, lai izvadītu visu informāciju uzreiz, zemāk redzamas dažas no funkcijām.

Sākumā var apskatīt visus reģistrētos lietotājus:

20.att “Visi lietotāji”



4.kods”Visu lietotāju parādīšanas kods”

// Fetch all users

app.get("/all-users", (req, res) => {

  const sql\_query = 'SELECT \* FROM users';

  connection.query(sql\_query, (err, result) => {

    if (err) throw err;

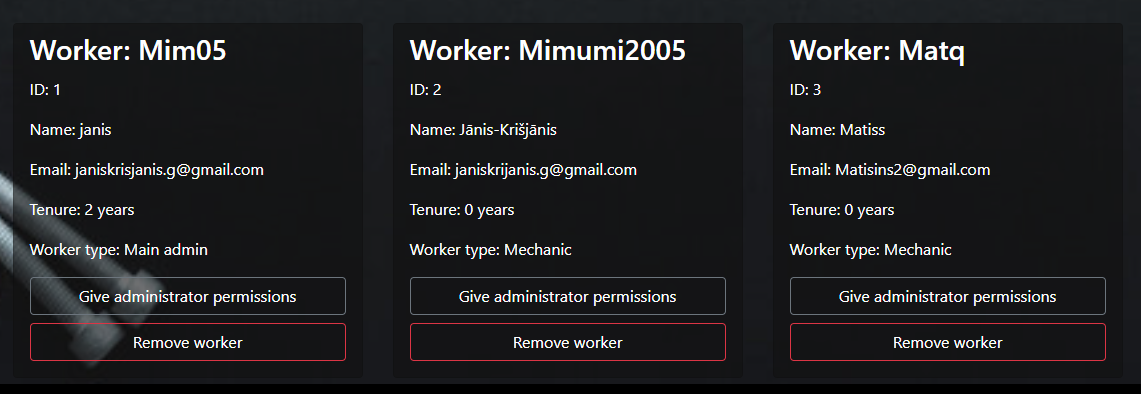
    res.send(result);

  });

});

Tad ar papildus query, iespējams redzēt tikai tos lietotājus kuri ir darbinieki, bet jārāda arī viņu konta informāciju, tapēc tiek apvienotas šīs tabuls un rādīta papildus informācija:

21. att “Darbinieki”



5. kods “Darbinieku rādīšanas kods”

// Fetch all worker info

app.post("/all-workers", (req, res) => {

  // Query to fetch all idUser values from the Workers table

  const workerQuery = 'SELECT idUser FROM Workers';

  connection.query(workerQuery, (err, workerResults) => {

    if (err) {

      console.error('Error querying Workers table:', err);

      return res.status(500).json({ message: 'Error fetching worker data' });

    }

    // Array to store combined user and worker information

    const combinedData = [];

    // Loop through each workerResult to fetch combined user and worker information

    workerResults.forEach(worker => {

      const idUser = worker.idUser;

      // Query to fetch combined information for the current idUser

      const combinedQuery = 'SELECT Users.\*, Workers.\* FROM Users INNER JOIN Workers ON Users.idUser = Workers.idUser WHERE Users.idUser = ?';

      connection.query(combinedQuery, [idUser], (err, combinedResult) => {

        if (err) {

          console.error(`Error querying combined information for idUser ${idUser}:`, err);

          return;

        }

        // Push combined information to the combinedData array

        combinedData.push(...combinedResult);

        // If all combined data is fetched, send the response

        if (combinedData.length === workerResults.length) {

          res.json(combinedData);

        }

      });

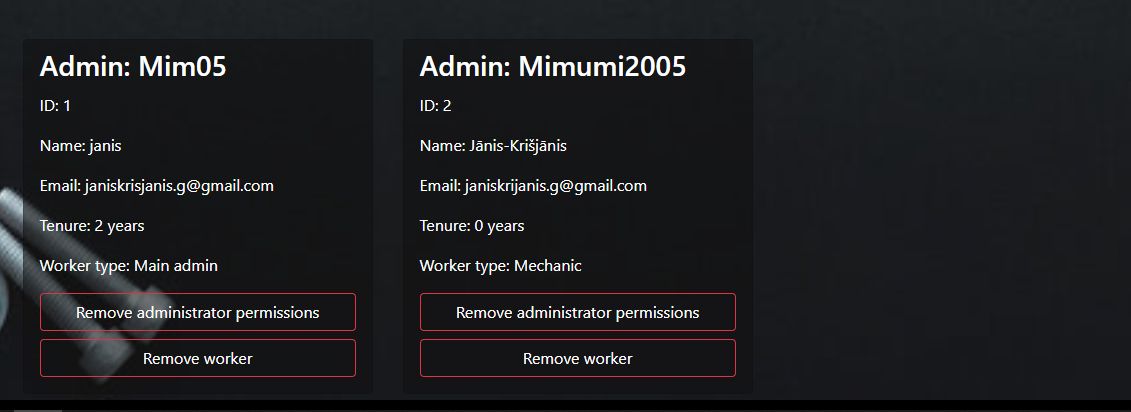
    });

  });

});

Pēc tās pašas loģikas iespējams arī apskatīt tikai administratorus:

22.att “Tikai administratori”



6. Kods “Tikai administratoru izvades kods”

// Fetch all admin info

app.post("/all-admins", (req, res) => {

  // Query to fetch all idUser values from the administrators table

  const adminQuery = 'SELECT idUser FROM administrators';

  connection.query(adminQuery, (err, adminResults) => {

    if (err) {

      console.error('Error querying Workers table:', err);

      return res.status(500).json({ message: 'Error fetching worker data' });

    }

    // Array to store combined user and worker information

    const combinedData = [];

    // Loop through each workerResult to fetch combined user and worker information

    adminResults.forEach(admin => {

      const idUser = admin.idUser;

      // Query to fetch combined information for the current idUser

      const combinedQuery = 'SELECT Users.\*, Workers.\* FROM Users INNER JOIN Workers ON Users.idUser = Workers.idUser WHERE Users.idUser = ?';

      connection.query(combinedQuery, [idUser], (err, combinedResult) => {

        if (err) {

          console.error(`Error querying combined information for idUser ${idUser}:`, err);

          return;

        }

        // Push combined information to the combinedData array

        combinedData.push(...combinedResult);

        // If all combined data is fetched, send the response

        if (combinedData.length === adminResults.length) {

          res.json(combinedData);

        }

      });

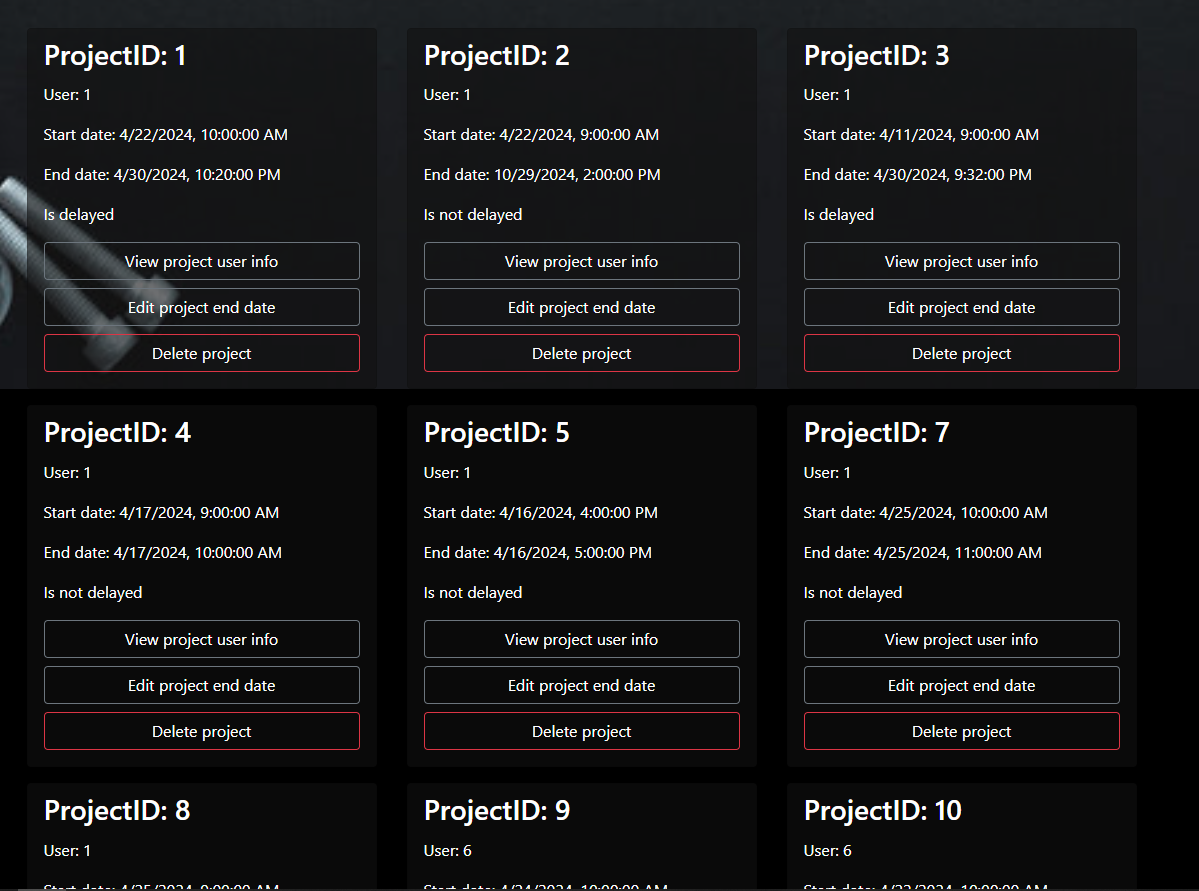
    });

  });

});

Pēc līdzīgas loģikas ir iespējams arī apskatīt projektus, tos var apskatīt gan darbinieki, gan administratori tapēc būs 2 veidi kā izskatīties šie dati uz ekrāna:

23. att “Visi projekti”



7. kods “Visu projektu izvades kods”

app.get("/all-projects", (req, res) => {

  const sql\_query = 'SELECT \* FROM projects';

  connection.query(sql\_query, (err, result) => {

    if (err) throw err;

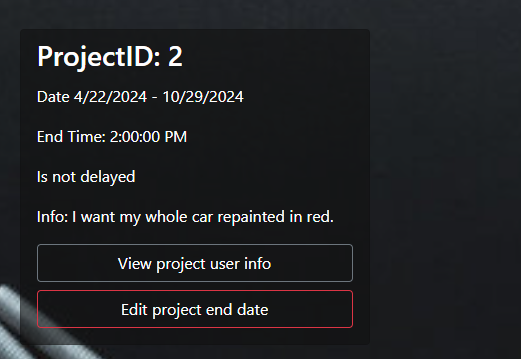
    res.send(result);

  });

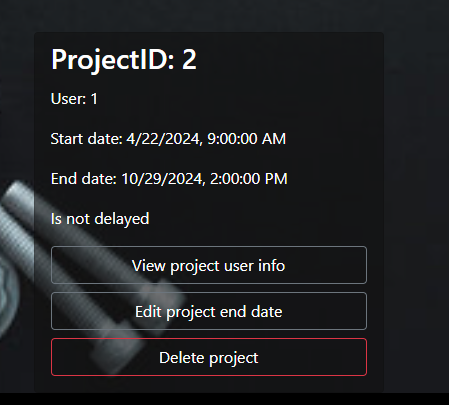
});

Tad iespējams atdalīt tikai aktīvos projektus, tas ir, tie projekti, kuru sākšanās datums ir sasniegts un beigšanās datums arī nav sasniegts:

24. att “Aktīvie projekti – darbinieku skats”



25. att “Aktīvie projekti – Administratora skats”



8. kods “Aktīvo projektu izvades kods”

app.get("/active-projects", (req, res) => {

  const curdate = new Date();

  const isoStringDate = curdate.toISOString();

  const sql\_query = `SELECT \* FROM projects WHERE '${isoStringDate}' BETWEEN StartDate AND EndDateProjection`;

  connection.query(sql\_query, (err, result) => {

    if (err) throw err;

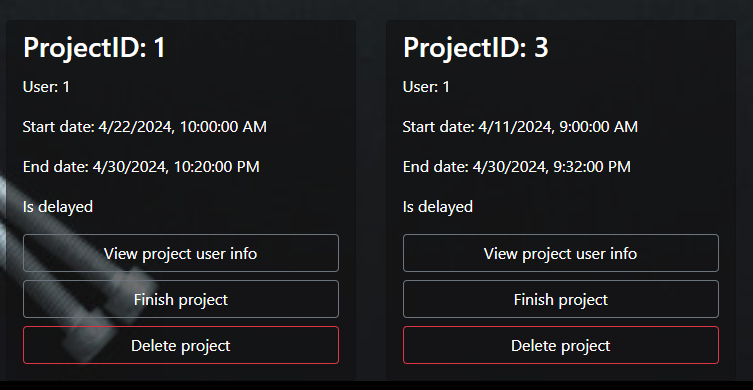
    res.send(result);

  });

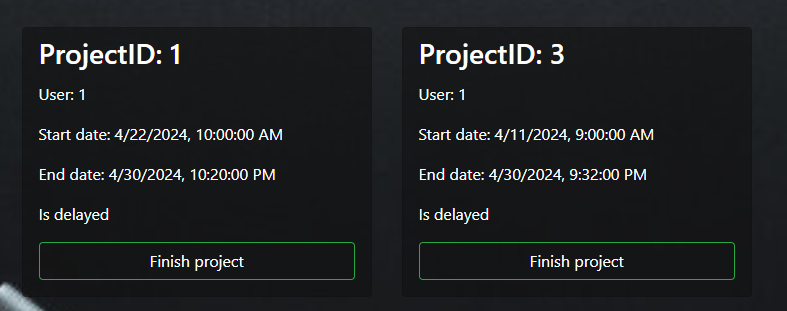
});

Vēl var apskatīt tikai “Delayed” projektus, kuriem “Delayed” vērtība ir “TRUE”, jo šiem projektiem nepieciešama papildus darbība, tos var pabeigt tikai darbiniekam vai administratoram izvēloties pogu “Finish Project”:

26. att “Administratoriem- ”Delayed” projektu izvade”



27. att “Darbiniekiem- ”Delayed” projektu izvade”



9. kods “Delayed projektu izvades kods”

// Fetch active projects (endDate time is not yet reached)

app.get("/delayed-projects", (req, res) => {

  const sql\_query = 'SELECT \* FROM projects WHERE `Delayed` = 1';

  connection.query(sql\_query, (err, result) => {

    if (err) throw err;

    console.log(result);

    res.send(result);

  });

});

## Datu integritātes nodrošināšanas metožu izmantošana

### Lietotāju autentifikācija

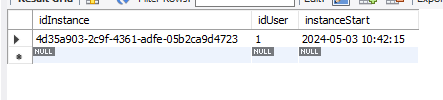
Lai sistēma spētu zināt, kad nepieciešams lietotājam rādīt tikai administratoriem vai darbiniekiem domātās pogas tiek izmantoti gan datu bāzes “user\_instance” tabulas dati, gan mājas lapas “cookies” glabātā informācija, kurā tiek saglabāts vai lietotājam nepieciešamas šīs papildus pogas.

29. att “Cookies saglabātā informācija”



Izmantojot šo informāciju, kopā ar datu bāzē glabāto informāciju par katra lietotāja pieslēgšanās gadījumu:

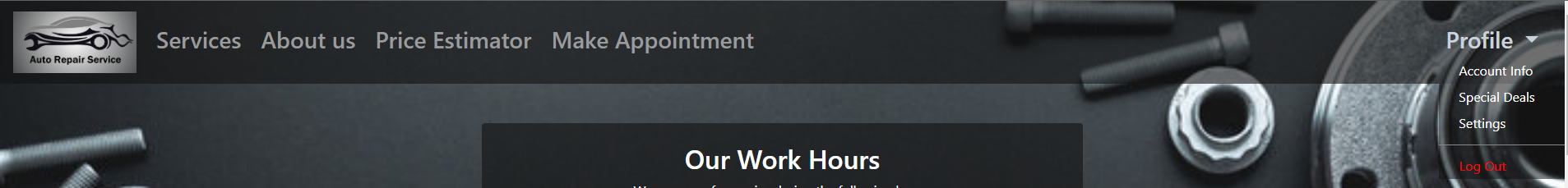
16. att “Datu bāzes glabātā informācija par pieslēgšanās gadījumu”



Varam rādīt dažādu informāciju lietotājiem kas ir darbinieki vai administrator, zemāk būs redzami ekrānuzņēmumi no tām pašām pogu sadaļām, bet mainīti lietotāju konti no parasta lietotāja, uz darbinieka un tad administratora.

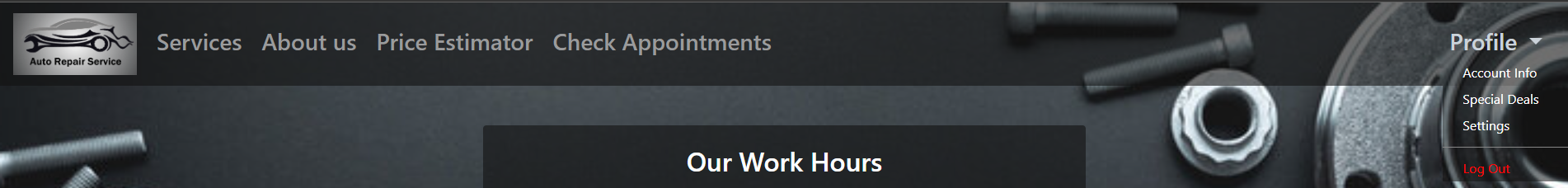
Parasts lietotājs:

30. att “Parasta lietotāja navbar”



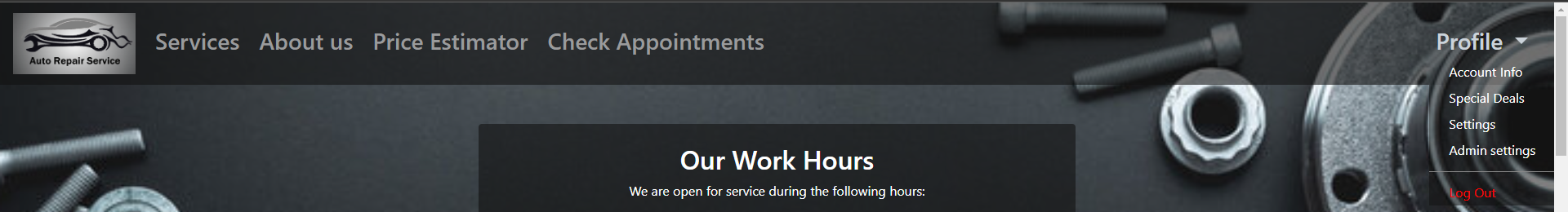
Darbinieks (“Make Appointment” tiek aizstāts ar “Check Appointments”):

31. att “Darbinieka navbar”



Administrators (papildus redzams “Admin settings” zem “Profile” apakš pogām):

32. att “Administratora navbar”



### Funkcionālo pienākumu sadalīšanā

Kā jau minēts iepriekšējajā punktā, lai jebkurš nespētu darboties kā administrators, tiek pārbaudīta katra lietotāja pieslēgšanās gadījums un pievienotas pogas, kuras ļauj tālāk darboties kā administratoram, tikai lietotājiem, kuri jau pārbaudīti kā administrator gan “Cookies”, gan pašā datu bāzē, pārbaudot vai pieslēgšanās gadījums saistīts ar kontu kurš ir administrators vai nē.

# NOBEIGUMS

Kopumā ir izveidota sistēma, kurā ir gan datu bāze, gan mājaslapa, kurām sadarbojoties ir iespējams veikt dažādas darbības. Veidot profilu, rediģēt šo kontu, pieteikties remontam, kā arī administratora iespējas darboties ar datu bāzes datiem un darbiniekiem iespēja apskatīt viņiem nepieciešamos datus, tas viss iespējams caur mājaslapu.

Veidojot projektu bija dažādas problēmām, nācās arī izmainīt datu bāzes struktūru, lai tā sakristu ar to kāda bija vajadzīga, bet kopumā gāja raiti un ja radās problēmas bija iespējams tās atrisināt samērā viegli.

Projektu, protams, vēl iespējams uzlabot, pats vēlētos pievienot dažādas citas funkcionalitātes un citādu uzlabot to, piemēram, pievienot gaišo režīmu, paroles maiņu caur e-pastu, ēpasta ziņu funkcionalitāti, papildus drošības uzlabošanu un citas svarīgas funkcijas.

# INFORMĀCIJAS AVOTI

<https://www.mysqltutorial.org/mysql-basics/mysql-generated-columns/>

<https://dev.mysql.com/doc/refman/8.0/en/windows-start-command-line.html>

<https://stackoverflow.com/questions/12197053/express-cookie-return-undefined> (un citas stackoverflow.com sadaļas)

<https://www.youtube.com/watch?v=7S_tz1z_5bA>

<https://www.dolthub.com/blog/2023-10-27-uuid-keys/#:~:text=To%20automatically%20generate%20a%20UUID,merges%20on%20branches%20and%20clones>

<https://www.dribbble.com>

<https://www.freecodecamp.org/news/authenticate-users-node-app/>

<https://www.youtube.com/watch?v=AuHNCbnQHBc>

<https://e.rvt.lv/course/view.php?id=308&section=2>

# PIELIKUMS

### Datu bāzes kods:

CREATE DATABASE IF NOT EXISTS `carrepairshop` /\*!40100 DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci \*/ /\*!80016 DEFAULT ENCRYPTION='N' \*/;

USE `carrepairshop`;

-- MySQL dump 10.13 Distrib 8.0.36, for Win64 (x86\_64)

--

-- Host: localhost Database: carrepairshop

-- ------------------------------------------------------

-- Server version 8.0.36

/\*!40101 SET @OLD\_CHARACTER\_SET\_CLIENT=@@CHARACTER\_SET\_CLIENT \*/;

/\*!40101 SET @OLD\_CHARACTER\_SET\_RESULTS=@@CHARACTER\_SET\_RESULTS \*/;

/\*!40101 SET @OLD\_COLLATION\_CONNECTION=@@COLLATION\_CONNECTION \*/;

/\*!50503 SET NAMES utf8 \*/;

/\*!40103 SET @OLD\_TIME\_ZONE=@@TIME\_ZONE \*/;

/\*!40103 SET TIME\_ZONE='+00:00' \*/;

/\*!40014 SET @OLD\_UNIQUE\_CHECKS=@@UNIQUE\_CHECKS, UNIQUE\_CHECKS=0 \*/;

/\*!40014 SET @OLD\_FOREIGN\_KEY\_CHECKS=@@FOREIGN\_KEY\_CHECKS, FOREIGN\_KEY\_CHECKS=0 \*/;

/\*!40101 SET @OLD\_SQL\_MODE=@@SQL\_MODE, SQL\_MODE='NO\_AUTO\_VALUE\_ON\_ZERO' \*/;

/\*!40111 SET @OLD\_SQL\_NOTES=@@SQL\_NOTES, SQL\_NOTES=0 \*/;

--

-- Table structure for table `administrators`

--

DROP TABLE IF EXISTS `administrators`;

/\*!40101 SET @saved\_cs\_client = @@character\_set\_client \*/;

/\*!50503 SET character\_set\_client = utf8mb4 \*/;

CREATE TABLE `administrators` (

`idUser` int NOT NULL,

`AdminTenure` int DEFAULT NULL,

PRIMARY KEY (`idUser`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

/\*!40101 SET character\_set\_client = @saved\_cs\_client \*/;

--

-- Table structure for table `projects`

--

DROP TABLE IF EXISTS `projects`;

/\*!40101 SET @saved\_cs\_client = @@character\_set\_client \*/;

/\*!50503 SET character\_set\_client = utf8mb4 \*/;

CREATE TABLE `projects` (

`idProjects` int NOT NULL AUTO\_INCREMENT,

`idUser` int NOT NULL,

`StartDate` varchar(45) DEFAULT NULL,

`EndDateProjection` varchar(45) DEFAULT NULL,

`Delayed` tinyint DEFAULT '0',

`ProjectInfo` varchar(255) DEFAULT NULL,

PRIMARY KEY (`idProjects`),

KEY `idUser\_idx` (`idUser`),

CONSTRAINT `user` FOREIGN KEY (`idUser`) REFERENCES `users` (`idUser`)

) ENGINE=InnoDB AUTO\_INCREMENT=15 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

/\*!40101 SET character\_set\_client = @saved\_cs\_client \*/;

--

-- Table structure for table `user\_instance`

--

DROP TABLE IF EXISTS `user\_instance`;

/\*!40101 SET @saved\_cs\_client = @@character\_set\_client \*/;

/\*!50503 SET character\_set\_client = utf8mb4 \*/;

CREATE TABLE `user\_instance` (

`idInstance` varchar(36) NOT NULL,

`idUser` int DEFAULT NULL,

`instanceStart` datetime DEFAULT NULL,

PRIMARY KEY (`idInstance`),

KEY `user\_logged\_idx` (`idUser`),

CONSTRAINT `user\_logged` FOREIGN KEY (`idUser`) REFERENCES `users` (`idUser`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

/\*!40101 SET character\_set\_client = @saved\_cs\_client \*/;

--

-- Table structure for table `users`

--

DROP TABLE IF EXISTS `users`;

/\*!40101 SET @saved\_cs\_client = @@character\_set\_client \*/;

/\*!50503 SET character\_set\_client = utf8mb4 \*/;

CREATE TABLE `users` (

`idUser` int NOT NULL AUTO\_INCREMENT,

`Name` varchar(15) DEFAULT NULL,

`Username` varchar(20) DEFAULT NULL,

`Email` varchar(45) DEFAULT NULL,

`password` varchar(45) DEFAULT NULL,

PRIMARY KEY (`idUser`)

) ENGINE=InnoDB AUTO\_INCREMENT=8 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

/\*!40101 SET character\_set\_client = @saved\_cs\_client \*/;

--

-- Table structure for table `workers`

--

DROP TABLE IF EXISTS `workers`;

/\*!40101 SET @saved\_cs\_client = @@character\_set\_client \*/;

/\*!50503 SET character\_set\_client = utf8mb4 \*/;

CREATE TABLE `workers` (

`idUser` int NOT NULL,

`tenure` int DEFAULT NULL,

`StartWorkDate` date DEFAULT NULL,

`WorkerType` varchar(45) DEFAULT NULL,

PRIMARY KEY (`idUser`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

/\*!40101 SET character\_set\_client = @saved\_cs\_client \*/;

/\*!40103 SET TIME\_ZONE=@OLD\_TIME\_ZONE \*/;

/\*!40101 SET SQL\_MODE=@OLD\_SQL\_MODE \*/;

/\*!40014 SET FOREIGN\_KEY\_CHECKS=@OLD\_FOREIGN\_KEY\_CHECKS \*/;

/\*!40014 SET UNIQUE\_CHECKS=@OLD\_UNIQUE\_CHECKS \*/;

/\*!40101 SET CHARACTER\_SET\_CLIENT=@OLD\_CHARACTER\_SET\_CLIENT \*/;

/\*!40101 SET CHARACTER\_SET\_RESULTS=@OLD\_CHARACTER\_SET\_RESULTS \*/;

/\*!40101 SET COLLATION\_CONNECTION=@OLD\_COLLATION\_CONNECTION \*/;

/\*!40111 SET SQL\_NOTES=@OLD\_SQL\_NOTES \*/;

-- Dump completed on 2024-05-03 11:47:54