

**JĒKABPILS TEHNOLOĢIJU TEHNIKUMS  
PROGRAMMĒŠANA  
PROGRAMMĒŠANAS TEHNIĶIS**

**Rubika kuba risināšanas analizators**

Kvalifikācijas darbs

Darba autors

\_\_\_\_\_  
(paraksts, datums)

Kristijans Vēveris  
4.p. kurss

Darba vadītājs

\_\_\_\_\_  
(paraksts, datums)

Dina Valpētere

Jēkabpils  
2025

## SATURS

|   |    |
|---|----|
| ATTĒĻU SARAKSTS .....                                       | 3  |
| DARBĀ LIETOTO SAĪSINĀJUMU SARAKSTS .....                    | 4  |
| IEVADS .....  | 5  |
| 1. PROGRAMMAS PRODUKTA PROJEKTĒŠANA .....                   | 6  |
| 1.1. Funkcionālo prasību apraksts .....                     | 6  |
| 1.2. Nefunkcionālo prasību apraksts.....                    | 13 |
| 2. PROGRAMMAS PRODUKTA RISINĀŠANAS LĪDZEKĻU IZVĒLE.....     | 15 |
| 2.1. Programmas produkta izstrādes līdzekļu izvēle .....    | 15 |
| 2.2. Programmas produkta izstrādes tehnoloģiju izvēle ..... | 16 |
| 3. PROGRAMMAS PRODUKTA REALIZĀCIJA .....                    | 18 |
| 3.1. Risināšanas metožu un algoritmu realizācija.....       | 18 |
| 3.2. Saskarnes realizācija.....                             | 21 |
| 3.3. Programmas produkta atbilstība specifikācijai.....     | 26 |
| 4. PROGRAMMAS PRODUKTA DATU STRUKTŪRAS APRAKSTS .....       | 32 |
| 4.1. Index tabulas.....                                     | 32 |
| 4.2. Algoritmu tabulas .....                                | 33 |
| 4.3. Lietotāji.....   | 33 |
| 4.4. Lietotāja Algoritmi .....                              | 34 |
| 4.5. Klašu diagramma .....                                  | 34 |
| 4.6. ENUM Edge un Corner klases .....                       | 35 |
| 4.7. CubieCube klase .....                                  | 35 |
| 4.8. Tableloader klase.....                                 | 37 |
| 4.9. IDA_Star_F2L un IDA_Star_Cross klases.....             | 37 |
| 4.10. RubikCube klase .....                                 | 38 |
| 5. PROGRAMMAS PRODUKTA LIETOTĀJA CEĻVEDIS .....             | 40 |
| 5.1. Reģistrācija/ Pierakstīšanās .....                     | 40 |
| 5.2. Risinājumu meklēšana.....                              | 40 |
| 5.3. Risinājumu filtrēšana.....                             | 42 |
| 5.4. Algoritmu preferenču filtrēšana.....                   | 44 |
| 5.5. Algoritmu preferenču pievienošana.....                 | 45 |
| 5.6. Algoritmu preferenču dzēšana.....                      | 45 |
| SECINĀJUMI .....  | 47 |
| IZMANTOTĀS INFORMĀCIJAS AVOTU SARAKSTS.....                 | 48 |
| PIELIKUMI.....  | 49 |

## ATTĒLU SARAKSTS

|   |    |
|---|----|
| 1. attēls. Datu plūsmas diagrama .....                    | 19 |
| 2. attēls. Pieslēgšanās/reģistrācijas lapas saskarne..... | 21 |
| 3. attēls. Galvenās risināšanas lapas saskarne .....      | 22 |
| 4. attēls. Lietotāju preferenču saskarne .....            | 23 |
| 5. attēls. Filtrēšanas modāļa saskarne .....              | 24 |
| 6. attēls. ER diagrama.....                               | 32 |
| 7. attēls. Index tabulas .....                            | 32 |
| 8. attēls. Algoritmu tabulas.....                         | 33 |
| 9. attēls. Lietotāju tabula.....                          | 33 |
| 10. attēls. Lietotāju preferenču tabulas .....            | 34 |
| 11. attēls. Kļāšu diagrammas enum klasēm.....             | 35 |
| 12. attēls. CubieCube klases diagrama .....               | 35 |
| 13. attēls. Tableloader klases diagrama .....             | 37 |
| 14. attēls. IDA* algoritmu klases diagramas .....         | 37 |
| 15. attēls. JavaScript Cube klases diagrama .....         | 38 |
| 16. attēls. Pieslēgšanās/Reģistrācijas konteineris .....  | 40 |
| 17. attēls. Maisījuma lauks .....                         | 40 |
| 18. attēls. Risinātāja sākšanas poga .....                | 41 |
| 19. attēls. Progresu konteineru sekcija.....              | 41 |
| 20. attēls. Risinājuma kartiņa .....                      | 42 |
| 21. attēls. Risinājumu sekcija .....                      | 43 |
| 22. attēls. Filtru modālis .....                          | 44 |
| 23. attēls. Algoritmu preferenču izvēles poga .....       | 44 |
| 24. attēls. Algoritmu preferenču modālis .....            | 45 |
| 24. attēls. Saglabāt algoritmu pogas validācija .....     | 45 |
| 25. attēls. Saglabātais algoritms .....                   | 46 |

## **DARBĀ LIETOTO SAĪSINĀJUMU SARAKSTS**

WCA – World Cubing Association

F2L – First 2 Layers

OLL – Orientation of the Last Layer

PLL – Permutation of the Last Layer

CFOP – Cross F2L OLL PLL

IDA\* - Iterative Deepening A\*

## IEVADS

Rubika kuba risināšanas optimizācija ir nozīmīga gan no teorētiskā, gan praktiskā viedokļa. Kaut arī kuba risināšana ir populārs izaicinājums gan amatieriem, gan profesionāļiem, efektīvu algoritmu izstrāde joprojām ir aktuāls uzdevums datu struktūru un meklēšanas algoritmu pētniecībā.

IDA\* algoritma izmantošana CFOP metodes optimizācijai ir inovatīvs risinājums, jo tas apvieno meklēšanas tehnikas ar iepriekš aprēķinātām datubāzēm, kas varētu būt nozīmīgs progress kuba risināšanas algoritmu jomā. Šāda sistēma var sniegt praktisku ieguvumu gan sacensību dalībniekiem, kuri vēlas uzlabot savu risināšanas laiku, gan arī programmatūras izstrādātājiem, kas interesējas par heuristisko meklēšanas algoritmu efektivitātes uzlabošanu.

Pienesums un nozīmība Rubika kuba risināšanas optimizācijas algoritms var sniegt nozīmīgu pienesumu vairākās jomās

1. Ātrie risinājumi: Sacensību dalībnieki var izmantot algoritma datus, lai trenētos un optimizētu savus risinājumus.
2. Izglītība un pētniecība: Datubāžu un meklēšanas algoritmu analīze ir nozīmīga studentiem un pētniekiem, kas interesējas par datorzinātnēm un mākslīgo intelektu.
3. Datu struktūru un algoritmu pielietojums: Sistēma var kalpot kā labs prakses piemērs efektīvas meklēšanas algoritmu izmantošanā, kas var tikt pāradaptēti citām jomām, piemēram, robotikā vai loģistikā.

Darba autors ir ilgstoši interesējies par algoritmu izstrādi un to pielietojumu praktisku problēmu risināšanā. Rubika kuba risināšana un sacensību stratēģiju izstrāde ir viena no autora aizrautībām, kas veicināja interesi par efektīvu risināšanas metožu optimizāciju.

Darba mērķis ir izstrādāt un optimizēt Rubika kuba risināšanas algoritmu, balstoties uz CFOP metodi un IDA\* algoritma pielietojumu, izmantojot datubāzes, lai paātrinātu risinājumu aprēķinus.

Darbā veicamie uzdevumi:

1. Definēt Rubika kuba risināšanas sistēmas funkcionalitātes prasības.
2. Noteikt veikspējas, drošības, lietojamības un elastības kritērijus.
3. Izstrādāt sistēmas arhitektūru.
4. Projektēt un realizēt datubāzes risinājumu OLL un PLL soļu glabāšanai.
5. Izveidot sasaisti starp IDA\* algoritmu un datubāzi.
6. Implementēt IDA\* algoritmu un veikt koda refaktorēšanu.
7. Izstrādāt lietotāja ceļvedi pamatfunkcionalitātes izmantošanas aprakstam.

# 1. PROGRAMMAS PRODUKTA PROJEKTĒŠANA

## 1.1. Funkcionālo prasību apraksts

### 1.1.1. FP.001: Risinājumu meklēšana

| FP.001   | Risinājumu meklēšanas | Obligāta |
|--|-----------------------|----------|
| Ievads   |                       |          |
| <p>Izmantojot klaviatūru ievada Rubik's kuba maisījumu izmantojot WCA noteikumos noteiktās notācijas.</p> <p>Šī funkcija ir galvenā risinātājas funkcija, kas atgriež lietotājam vairākus derīgus optimālus risinājumus attiecīgumam kuba maisījumam pierakstītam WCA notācijā.</p>  |                       |          |
| Ievade   |                       |          |
| String datu tipa maisījums WCA notācijā.   |                       |          |
| Apstrāde   |                       |          |
| <ul style="list-style-type: none"><li>Izmantojot ievadīto maisījumu, tiek izveidots virtuālais kubs, kas skaitīsies kā sākumstadija risinājuma algoritmam.</li><li>Atrod Cross risinājumu, izmantojot IDA* algoritmu.</li><li>Atrod F2L risinājumu, izmantojot IDA* algoritmu.</li><li>Atbilstošajam F2L risinājumam atrod risinājumu priekš OLL fāzes izmantojot datu bāze saglabātos risinājumus.</li><li>Atbilstošajam OLL risinājumam atrod risinājumu priekš PLL fāzes izmantojot datu bāze saglabātos risinājumus.</li></ul> |                       |          |
| Izvade   |                       |          |
| Lietotājam tiek atgriezti vairāki optimāli risinājumi ievadītajam maisījumam.  |                       |          |

### 1.1.2. FP.002: Risinājumu filtri

| FP.002   | Risinājumu filtri | Obligāta |
|--|-------------------|----------|
| Ievads   |                   |          |
| <p>Lietotājs var pielāgot un filtrēt piedāvātos Rubika kuba risinājumus pēc dažādiem kritērijiem, kas ļauj atlasīt vēlamo risinājuma tipu vai stilistiku (piemēram, tikai ar OLL skip, vai izmantojot paša definētus algoritmus).</p>                        |                   |          |
| Ievade   |                   |          |
| <ul style="list-style-type: none"><li>Kārtošanas filtri:<ul style="list-style-type: none"><li>Pēc kopējā risinājuma garuma (Īsākais → garākais vai garākais → īsākais)</li><li>Pēc OLL algoritma garuma</li><li>Pēc PLL algoritma garuma</li></ul></li></ul> |                   |          |

|  |
|--|
| <ul style="list-style-type: none"> <li>• Izslēgšanas/iekļaušanas filtri: <ul style="list-style-type: none"> <li>○ Tikai ar OLL skip</li> <li>○ Tikai ar PLL skip</li> <li>○ Tikai ar LL skip (OLL + PLL skip)</li> <li>○ Tikai ar lietotāja definētiem OLL algoritmiem</li> <li>○ Tikai ar lietotāja definētiem PLL algoritmiem</li> <li>○ Tikai ar lietotāja definētiem OLL un PLL algoritmiem</li> </ul> </li> </ul>   |
| <b>Apstrāde</b>  |
| <ul style="list-style-type: none"> <li>• Visi iepriekš ģenerētie risinājumi tiek analizēti saskaņā ar filtriem.</li> <li>• Tiek atlasīti tikai tie risinājumi, kuri atbilst konkrētajiem filtru nosacījumiem.</li> <li>• Risinājumu saraksts tiek pārkārtots atbilstoši kārtšanas filtriem.</li> <li>• Ja piemērots lietotāja definēto algoritmu filtrs, tad pārbauda, vai risinājumos izmantotie OLL/PLL algoritmi sakrīt ar lietotāja saglabāto algoritmu sarakstu.</li> <li>• Nepiemērotie risinājumi tiek noņemti no rezultātu kopas.</li> </ul> |
| <b>Izvade</b>  |
| <ul style="list-style-type: none"> <li>• Lietotājam tiek atgriezta tikai tā risinājumu kopa, kas pilnībā atbilst izvēlētajiem filtriem.</li> <li>• Kārtība sarakstā atbilst izvēlētajai kārtšanai (piemēram, no īsākā uz garāko vai pēc PLL fāzes garuma).</li> <li>• Ja neviens risinājums neatbilst filtriem, tiek parādīts atbilstošs paziņojums (piem., "Nav atrasts neviens risinājums, kas atbilst izvēlētajiem filtriem").</li> </ul>   |

### 1.1.3. FP.003: Vēlamo algoritmu filtri

| FP.003   | Vēlamo algoritmu filtri | Obligāta |
|--|-------------------------|----------|
| <b>Ievads</b>  |                         |          |
| <p>Lietotājs var norādīt, kā izmantot viņa saglabātos OLL/PLL algoritmus risinājumu ģenerēšanas laikā. Šī funkcionalitāte pieejama tikai reģistrētiem un pieslēgušiem lietotājiem, kuriem vismaz viens OLL vai PLL algoritms ir saglabāts kā vēlamais algoritms.</p> |                         |          |
| <b>Ievade</b>  |                         |          |
| <p>No lietotāja tiek saņemta viena no šādām izvēlēm:</p> <ul style="list-style-type: none"> <li>• <b>Use default algorithms</b></li> </ul> <p>Tiek izmantota standarta algoritmu datubāze (ignorējot lietotāja saglabātās preferences).</p>                          |                         |          |

|   |
|---|
| <ul style="list-style-type: none"> <li>• <b>Prioritize my algorithms</b><br/>Risinājumu ģenerēšanas laikā tiek prioritāri izmantoti lietotāja saglabātie algoritmi, ja tie atbilst konkrētajai situācijai.</li> <li>• <b>Use only my algorithms</b><br/>Ģenerē risinājumus tikai tad, ja attiecīgajai fāzei ir pieejams lietotāja definēts algoritms (OLL, PLL). Ja šāds nav pieejams, attiecīgajam risinājumam izmanto datu bāzē saglabātos algoritmus</li> </ul>  |
| Apstrāde  |
| <ul style="list-style-type: none"> <li>• Pirms algoritmu preferenču piemērošanas tiek pārbaudīts: <ul style="list-style-type: none"> <li>○ Vai lietotājs ir autentificējies sistēmā.</li> <li>○ Vai ir vismaz viens saglabāts OLL vai PLL algoritms.</li> </ul> </li> <li>• Balstoties uz izvēlēto režīmu: <ul style="list-style-type: none"> <li>○ <b>Use default algorithms:</b> algoritmi tiek atlasīti no sistēmas noklusējuma datubāzes.</li> <li>○ <b>Prioritize my algorithms:</b> meklē lietotāja algoritmu vispirms, ja nav atbilstoša – izmanto noklusējuma algoritmu.</li> <li>○ <b>Use only my algorithms:</b> meklē tikai lietotāja algoritmu; ja nav piemērota – risinājums netiek ģenerēts.</li> </ul> </li> <li>• Apstrāde tiek piemērota katram OLL un PLL risinājumam atsevišķi.</li> </ul> |
| Izvade  |
| Ģenerētie risinājumi atspoguļo izvēlēto algoritmu preferenču režīmu.  |

#### 1.1.4. FP.004: Lietotāja pieslēgšanas

| FP.004   | Lietotāja pieslēgšanas   | Obligāta |
|----------|--|----------|
| Ievads   | <p>Lietotājs ievada savu e-pasta adresi un paroli, lai pieslēgtos sistēmai. Ja ievadītais e-pasts vēl neeksistē sistēmā, automātiski tiek izveidots jauns konts, un lietotājs tiek pieslēgts. Ja konts jau eksistē, tiek pārbaudīta parole. Veiksmīgas pieslēgšanās gadījumā lietotājs tiek novirzīts uz galveno lapu.</p> |          |
| Ievade   | <ul style="list-style-type: none"> <li>• E-pasta adrese (String, obligāts)</li> <li>• Parole (String, obligāts)</li> </ul>   |          |
| Apstrāde | <p>Tiek pārbaudīts, vai e-pasta adrese eksistē sistēmā.</p> <ul style="list-style-type: none"> <li>• Ja neeksistē:</li> </ul>  |          |



|  |
|--|
| <ul style="list-style-type: none"> <li>○ Automātiski tiek mēģināts izveidot jaunu lietotāja kontu ar ievadīto e-pasta adresi un paroli.</li> <li>○ Lietotājs tiek autentificēts un novirzīts uz galveno lapu.</li> <li>• Ja eksistē: <ul style="list-style-type: none"> <li>○ Ja parole pareiza → lietotājs tiek autentificēts un novirzīts uz galveno lapu.</li> <li>○ Ja parole nepareiza → tiek parādīts paziņojums: “Nepareiza parole.”</li> </ul> </li> </ul> |
| Izvade   |
| <p>Pozitīvs scenārijs (veiksmīga pieslēgšanās vai konta izveide):</p> <p>Lietotājs tiek automātiski novirzīts uz galveno lapu.</p> <p>Negatīvs scenārijs (kļūdaina parole):</p> <p>Tiek parādīts kļūdas paziņojums “Nepareiza parole.” un pieslēgšanās netiek veikta.</p>  |

#### 1.1.5. FP.005: Lietotāja izrakstīšanās

| FP.005   | Lietotāja izrakstīšanās  | Obligāta |
|----------|--|----------|
| Ievads   | <p>Lietotājs var pārtraukt savu sesiju, nospiežot pogu "Izrakstīties". Šī darbība atvieno lietotāju no sistēmas un novērš piekļuvi personalizētām funkcijām (piemēram, algoritmu preferencēm, saglabātiem datiem u.c.).</p>  |          |
| Ievade   | <p>Nav specifiskas ievades no lietotāja puses, izņemot darbību izrakstīšanās pogas nospiešana.</p>   |          |
| Apstrāde | <ul style="list-style-type: none"> <li>• Tiek pārtraukta lietotāja autentifikācijas sesija (noņemti autorizācijas žetoni, sesijas dati vai sīkdatnes).</li> <li>• Lietotājs vairs nav identificēts kā pieslēgts sistēmai.</li> <li>• Tiek atiestatīts lietotāja interfeiss uz "neautentificēta lietotāja" stāvokli (piemēram, pazūd piekļuve algoritmu filtriem).</li> <li>• Pēc izrakstīšanās lietotājs tiek automātiski novirzīts uz pieslēgšanās lapu.</li> </ul> |          |
| Izvade   | <p>Lietotājs tiek pārdresēts uz pieslēgšanās lapu kā nepieslēgts lietotājs.</p> <p>Lietotājam vairs nav pieejamas funkcijas, kas pieejamas tikai pieslēgtiem lietotājiem.</p>  |          |

#### 1.1.6. FP.006: Saglabāt lietotāja OLL un PLL vēlamos algoritmus

| FP.006    Saglabāt lietotāja OLL un PLL vēlamos algoritmus    Obligāta  |
|---|
| Ievads  |
| Pieslēdzies lietotājs var norādīt un saglabāt savus personalizētos OLL un PLL algoritmus, kas tiks izmantoti risinājumu ģenerēšanā saskaņā ar izvēlētajām algoritmu preferencēm. Katram gadījumam (OLL/PLL) var pievienot vairāk nekā vienu algoritmu.  |
| Ievade  |
| Algoritma ievade: teksta lauks vienam algoritmam, WCA notācijā.   |
| Apstrāde  |
| <p>Autentifikācija:</p> <ul style="list-style-type: none"> <li>Funkcija pieejama tikai pieslēgtiem lietotājiem.</li> </ul> <p>Algoritma validācija:</p> <ul style="list-style-type: none"> <li>Sintakses pārbaude: tiek pārbaudīts, vai algoritms sastāv tikai no derīgām WCA kustībām (R, U, F, L, D, B, kopā ar ' un 2).</li> <li>Simulācija uz 3x3 kuba: <ul style="list-style-type: none"> <li>Tiek uzbūvēts atbilstošais OLL/PLL stāvoklis, kas atbilst izvēlētajam gadījumam.</li> <li>Uz šo gadījumu tiek piemērots lietotāja ievadītais algoritms.</li> <li>Tiek pārbaudīts, vai rezultāts ir pilnībā atrisināts kubs.</li> </ul> </li> <li>Ja algoritms neatrisina izvēlēto gadījumu, tas tiek noraidīts ar paziņojumu (piemēram, "Algoritms neatrisina izvēlēto gadījumu").</li> </ul> <p>Saglabāšana:</p> <ul style="list-style-type: none"> <li>Validēts algoritms tiek saglabāts lietotāja profilā zem konkrētā OLL vai PLL gadījuma.</li> <li>Ja šim gadījumam jau ir citi saglabāti algoritmi, jaunais tiek pievienots esošajam sarakstam.</li> <li>Lietotājs var jebkurā brīdī dzēst vai rediģēt savus algoritmus konkrētajam gadījumam.</li> </ul> |
| Izvade  |
| <p>Veiksmīgs scenārijs:</p> <p>Algoritms tiek pievienots sarakstam zem izvēlētās kartītes.</p> <p>Neveiksmīgs scenārijs:</p> <p>Poga "Pievienot" ir neaktīva, kamēr validācija nav izieta.</p>  |

#### 1.1.7. FP.007: Meklēt risinājumu pēc lietotāja Cross

| FP.007  | Meklēt risinājumu pēc lietotāja Cross | Vēlama |
|---|---------------------------------------|--------|
| Ievads  |                                       |        |
| Tā vietā lai lietotājam iedotu krusta risinājumus, lietotājs definē savu krusta risinājumu un risinātājs atrod optimizētus risinājumus atbilstošajam sajaukumam un krustam.   |                                       |        |
| Ievade  |                                       |        |
| Sajaukuma algoritma ievade: teksta lauks vienam algoritmam, WCA notācijā.<br>Krusta risinājuma algoritma ievade: teksta lauks vienam algoritmam, WCA notācijā.  |                                       |        |
| Apstrāde  |                                       |        |
| <ul style="list-style-type: none"> <li>Izmantojot ievadīto maisījumu, tiek izveidots virtuālais kubs, kas skaitīsies kā sākumstadija risinājuma algoritmam.</li> <li>Lietotāja ievadītais krusta risinājums tiek uzlikts uz virtuālā kuba un netiek meklēts krusta risinājums izmantojot IDA* algoritmu.</li> <li>Atrod F2L risinājumu, izmantojot IDA* algoritmu.</li> <li>Atbilstošajam F2L risinājumam atrod risinājumu priekš OLL fāzes izmantojot datu bāze saglabātos risinājumus.</li> <li>Atbilstošajam OLL risinājumam atrod risinājumu priekš PLL fāzes izmantojot datu bāze saglabātos risinājumus.</li> </ul> |                                       |        |
| Izvade  |                                       |        |
| Lietotājam tiek atgriezti vairāki optimāli risinājumi ievadītajam maisījumam.   |                                       |        |

#### 1.1.8. FP.008: Meklēšanas josla risinājumu sekcijā

| FP.008   | Meklēšanas josla risinājumu sekcijā | Vēlama |
|--|-------------------------------------|--------|
| Ievads   |                                     |        |
| Risinātājs var atgriezt vairākus tūkstošus risinājumus, meklēšanas opcija atvieglotu meklēt risinājumu, kas ir līdzīgs lietotāja risinājumam.  |                                     |        |
| Ievade   |                                     |        |
| Meklēšanas simbolu virkne: teksta lauks  |                                     |        |
| Apstrāde   |                                     |        |
| <ul style="list-style-type: none"> <li>Sistēma reāllaikā filtrē redzamos risinājumus, pamatojoties uz meklēšanas virkni.</li> <li>Meklēšana notiek pēc risinājuma algoritmiem, soļu skaita, un katra risinājuma īpatnībā (piemēram vai tas izlaiž OLL soli vai PLL soli).</li> </ul> |                                     |        |

|  |
|--|
| <ul style="list-style-type: none"> <li>Tiek izmantots daļējas atbilstības algoritms (piemēram, contains vai fuzzy match).</li> </ul> |
| Izvade   |
| Tiek atgriezti visi risinājumi, kas atbilst meklēšanas simbolu virknei.  |

#### 1.1.9. FP.009: Meklēšanas josla lietotāju preferenču lapā

| FP.009 : Meklēšanas josla lietotāju preferenču lapā   | Vēlams |
|---|--------|
| Ievads  |        |
| Kopā ir 57 OLL gadījumi un 21 PLL gadījumi un ievadīt katram gadījumam preferenci var būt sarežģīti, ja neiet pēc kārtas, tāpēc meklēšanas josla atvieglotu gadījuma meklēšanu  |        |
| Ievade  |        |
| Meklēšanas simbolu virkne: teksta lauks   |        |
| Apstrāde  |        |
| <ul style="list-style-type: none"> <li>Sistēma reāllaikā filtrē redzamos risinājumus, pamatojoties uz meklēšanas virkni.</li> <li>Meklēšana notiek pēc OLL vai PLL gadījuma nosaukuma (piemēram, OLL 1 vai PLL Aa)</li> <li>Tiek izmantots daļējas atbilstības algoritms (piemēram, contains vai fuzzy match).</li> </ul> |        |
| Izvade  |        |
| Tiek atgriezti visi gadījumi, kas atbilst meklēšanas simbolu virknei.   |        |

## 1.2. Nefunkcionālo prasību apraksts

### 1.2.1. NP.001: Veiktspēja

| Nr.p.k. | Prasību nosaukums               | Apraksts   | Obligāta / Vēlama |
|---------|---------------------------------|--|-------------------|
| 1.      | Ievadīto datu apstrādes ātrums  | Risinājumu ģenerēšanas laikam jābūt saprātīgam (optimāli <1 sekunde katram pārim) tipiskiem maisījumiem. | Obligāta          |
| 2.      | Atmiņas patēriņa efektivitāte   | Sistēmai jāizmanto atmiņas resursus efektīvi, optimizējot datu struktūras un kešatmiņu.                  | Obligāta          |
| 3.      | Procesora patēriņa efektivitāte | Risināšanas algoritmiem jābūt optimizētiem, lai nodrošinātu minimālu CPU noslodzi serverī.               | Obligāta          |

### 1.2.2. NP.002: Lietotāja pieredze

| Nr.p.k. | Prasību nosaukums | Apraksts   | Obligāta / Vēlama |
|---------|-------------------|--|-------------------|
| 1.      | Lietojamība       | Sistēmai jābūt intuitīvai un viegli lietojamai, nodrošinot lietotājiem vienkāršu piekļuvi visām funkcijām. | Obligāta          |
| 2.      | Pieejamība        | Sistēmai jābūt pieejamai pēc iespējas vairāk valodām, lai to var izmantot lietotāji visapkārt pasaulei.    | Vēlama            |

### 1.2.3. NP.003: Drošība un datu aizsardzība

| Nr.p.k. | Prasību nosaukums          | Apraksts   | Obligāta / Vēlama |
|---------|----------------------------|--|-------------------|
| 1.      | Lietotāja datu aizsardzība | Sistēmai jāglabā lietotāju konta dati (e-pasts, paroles, algoritmu preferenču dati) drošā veidā, izmantojot standarta šifrēšanas mehānismus. | Vēlama            |
| 2.      | Sesijas drošība            | Lietotāja sesijai jābūt aizsargātai pret nesankcionētu piekļuvi (piemēram, izmantojot drošus autentifikācijas žetonus un HTTPS).             | Vēlama            |

|    |                                    |  |        |
|----|------------------------------------|--|--------|
| 3. | Validācijas ievainojamību kontrole | Sistēmai jāaizsargājas pret kaitīgiem ievada datiem (piemēram, XSS, injekcijas), pārbaudot visu ievadi (īpaši algoritmu teksta lauku). | Vēlama |
|----|------------------------------------|--|--------|

#### 1.2.4. NP.004: Uzturamība un paplašināmība

| Nr.p.k. | Prasību nosaukums | Apraksts  | Obligāta / Vēlama |
|---------|-------------------|---|-------------------|
| 1.      | Moduļu struktūra  | Projekta kodam jābūt strukturētam modulāri, lai to būtu viegli paplašināt ar jaunām funkcionalitātēm. | Vēlama            |

#### 1.2.5. NP.003: Starp platformu saderība

| Nr.p.k. | Prasību nosaukums  | Apraksts  | Obligāta / Vēlama |
|---------|--------------------|---|-------------------|
| 1.      | Pārlūku saderība   | Sistēmai jādarbojas vismaz jaunākajās Chrome, Firefox un Safari versijās.                   | Obligāta          |
| 2.      | Mobilā pielāgotība | Sistēmai jābūt pielāgotai lietošanai arī uz viedtālruniem un planšetēm (responsive design). | Vēlama            |

## **2. PROGRAMMAS PRODUKTA RISINĀŠANAS LĪDZEKĻU IZVĒLE**

### **2.1. Programmas produkta izstrādes līdzekļu izvēle**

#### **2.1.1. Visual Studio Code**

Visual Studio Code (VS Code) ir moderna, viegla un jaudīga koda rediģēšanas un izstrādes vide, kas piedāvā daudzas funkcijas, kas atvieglo programmatūras izstrādi:

- Plašs paplašinājumu atbalsts – VS Code piedāvā tūkstošiem paplašinājumu, kas atvieglo darbu ar dažādām programmēšanas valodām un rīkiem.
- Integrēts Git atbalsts – ļauj ērti pārvaldīt versiju kontroli un sadarboties ar komandas biedriem.
- Debugging iespējas – nodrošina rīkus koda atklūdošanai un veiktspējas analīzei.
- Daudz platformu atbalsts – darbojas uz Windows, macOS un Linux, kas padara to elastīgu un pieejamu dažādiem lietotājiem.
- Viegls un ātrs – salīdzinājumā ar citām IDE, VS Code ir vieglāks un piedāvā augstu veiktspēju pat uz mazjaudīgiem datoriem.

#### **2.1.2. Draw.io**

Draw.io ir rīks vizuālo shēmu, diagrammu un struktūru veidošanai, kas ir būtisks programmatūras arhitektūras plānošanā un dokumentācijas izstrādē:

- Bez maksas un pieejams tiešsaistē – nav nepieciešams instalēt papildu programmatūru, var izmantot tiešsaistē vai lejupielādēt lokāli.
- Daudzveidīgas diagrammas – atbalsta UML diagrammas, ERD diagrammas, blokshēmas, plūsmas diagrammas un citus vizuālos attēlojumus.
- Integrācija ar citiem rīkiem – var saglabāt un koplietot diagrammas caur Google Drive, GitHub un citām platformām.
- Viegls un lietotājam draudzīgs interfeiss – ļauj ātri un ērti izveidot diagrammas bez sarežģītības.

#### **2.1.3. Windows Subsystem for Linux (WSL)**

WSL ir Microsoft izstrādāts rīks, kas ļauj lietotājiem darbināt Linux vidi tieši uz Windows sistēmas bez nepieciešamības pēc virtuālās mašīnas:

- Linux komandu rindu atbalsts – ļauj izmantot bash un citas Linux komandrindas rīkus Windows vidē.
- Integrācija ar Windows failu sistēmu – iespējams piekļūt Windows failiem no Linux vides un otrādi, kas atvieglo failu apmaiņu.

- Izstrāde un testēšana uz Linux bez atsevišķas mašīnas – ideāli piemērots izstrādātājiem, kuri vēlas testēt kodu Linux vidē, strādājot uz Windows.
- Atbalsta vairākas Linux distribūcijas – piemēram, Ubuntu, Debian, Fedora, Alpine u.c., kas ļauj izvēlēties sev piemērotāko vidi.
- WSL 2 ar reālu Linux kodolu – uzlabota versija, kas nodrošina labāku veiktspēju, pilnvērtīgāku saderību un Docker atbalstu.

#### **2.1.4. Figma**

Figma ir moderna, pārlūkā balstīta dizaina un prototipu veidošanas platforma, kas paredzēta saskarnes dizainam un sadarbībai starp dizaineriem un izstrādātājiem:

- Pārlūkā balstīta – nav jāinstalē – darbojas tieši pārlūkā, kas ļauj piekļūt no jebkuras ierīces bez lokālas instalācijas.
- Reāllaika sadarbība – vairāk lietotāju var vienlaicīgi strādāt pie viena dizaina, līdzīgi kā Google Docs.
- UI/UX dizains un prototipēšana – nodrošina rīkus lietotāja interfeisa veidošanai, animācijām, pārejām un interaktīvu prototipu izstrādei.
- Versiju kontrole un komentēšana – viegli izsekot izmaiņām un pievienot komentārus tieši dizainā, kas atvieglo komunikāciju starp komandām.
- Integrācija ar izstrādes rīkiem – iespējams eksportēt CSS, iOS un Android kodu fragmentus, kas paātrina izstrādes procesu.

## **2.2. Programmas produkta izstrādes tehnoloģiju izvēle**

### **2.2.1. C++**

C++ ir izvēlēta kā galvenā programmēšanas valoda risinātāja izstrādei, jo tā piedāvā augstu veiktspēju, zemu līmeņa kontroli pār atmiņu un plašas iespējas algoritmu optimizācijai. Tā ir ideāli piemērota skaitļošanas intensīviem uzdevumiem un nodrošina ātru izpildi.

### **2.2.2. CMake**

CMake tiek izmantots kā būvniecības rīks C++ projektam, jo tas nodrošina pārnēsājamību starp dažādām platformām un atvieglo koda pārvaldību, automātiski ģenerējot Makefile un citus nepieciešamos konfigūracijas failus.

### **2.2.3. Served**

Served ir viegls HTTP serveris, kas tiek izmantots kā starpposms starp dažādiem programmas komponentiem, nodrošinot efektīvu datu apmaiņu un API pieejamību.



#### **2.2.4. cubing.js**

cubing.js tiek izmantots Rubika kuba risinājumu vizualizācijai. Tā ir izvēlēta, jo nodrošina precīzu un interaktīvu kuba animāciju, kas ļauj lietotājam skaidri redzēt katra risinājuma soļa izpildi.

#### **2.2.5. Node.js**

Node.js tiek izmantots kā servera puses izpildvide, lai apstrādātu HTTP pieprasījumus, pārvaldītu API loģiku un koordinētu datu plūsmu starp klienta saskarni, risinātāju un datu glabātuvi. Node.js izvēlēts tā asinhronā darbības modeļa un augstās veiktspējas dēļ, kas padara to īpaši piemērotu sistēmām ar biežu datu apmaiņu un reāllaika funkcionalitāti. Tas arī ļauj izmantot vienotu JavaScript bāzi klienta un servera pusē, vienkāršojot izstrādi.

#### **2.2.6. Appwrite**

Appwrite tiek izmantots kā backend pakalpojums, kas nodrošina lietotāju autentifikāciju. Tas ļauj ātri integrēt tādas funkcijas kā e-pasta reģistrācija, pieteikšanās, sesiju pārvaldība bez nepieciešamības rakstīt backend no nulles.

#### **2.2.7. Axios**

Axios ir JavaScript bibliotēka, ko izmanto, lai veiktu HTTP pieprasījumus uz serveri. Tā atbalsta Promise balstītu arhitektūru, ļauj viegli nosūtīt GET, POST, PUT, DELETE pieprasījumus, kā arī apstrādāt atbildes un kļūdas vienkārši un skaidri.

#### **2.2.8. Cors**

CORS ir mehānisms, kas kontrolē, kā resursi tiek koplietoti starp dažādiem domēniem. Projektā tas tiek konfigurēts, lai ļautu frontend (piemēram, HTML lapai) piekļūt backend API (kas darbojas uz cita domēna vai porta), nepieļaujot drošības pārkāpumus.

#### **2.2.9. Express**

Express ir minimālistisks un elastīgs Node.js web ietvars, kas tiek izmantots API izveidei un servera loģikas realizēšanai. Tas ļauj definēt maršrutus, middleware un kontrolēt pieprasījumu/atbilžu plūsmu efektīvi un vienkārši.

#### **2.2.10. MongoDB Atlas**

MongoDB Atlas ir mākoņpakalpojums, kas nodrošina pilnībā pārvaldītu MongoDB datubāzi. Tas tiek izmantots datu glabāšanai (piemēram, lietotāju informācijai, risinājumiem utt.), nodrošinot augstu pieejamību, drošību un mērogojamību bez nepieciešamības pašam konfigurēt serverus.

### **3. PROGRAMMAS PRODUKTA REALIZĀCIJA**

#### **3.1. Risināšanas metožu un algoritmu realizācija**

##### **3.1.1. Lietotāja autentifikācijas process**

Lietotāja autentifikācijas process sākas, kad lietotājs ievada savus akreditācijas datus (e-pastu un paroli). Sistēma vispirms pārbauda, vai lietotājs jau eksistē datubāzē. Ja lietotājs neeksistē, process turpinās ar reģistrāciju, kurā vispirms tiek izveidots jauns lietotāja konts Appwrite sistēmā, izmantojot e-pastu, paroli un lietotājvārdu. Kad konts ir izveidots, lietotāja kolekcijā tiek izveidots jauns dokuments, kurā tiek saglabāta pamatinformācija par lietotāju. Šī kolekcija pastāv, lai autors vēlāk varētu paplašināt šo kolekciju, ja nepieciešams. (skatīt pielikumu Nr.1)

##### **3.1.2. IDA\* meklēšanas process**

###### **Algoritma pārskats**

Meklēšana tiek veikta ar IDA\* (Iterative Deepening A\*), kur iteratīvi tiek palielināta maksimālā pieļautā ceļa vērtība ( $f = g + h$ ). Katrā solī tiek novērtēts stāvoklis, balstoties uz vienkāršu heuristiku, lai izlemtu, vai ir vērts turpināt. Ja stāvoklis jau iepriekš apmeklēts mazākā dziļumā, tas tiek izlaists, pateicoties transponēšanas tabulai. Lai ierobežotu meklēšanas telpu, tiek izmantoti vienkārši atzarošanas nosacījumi, kas izslēdz nelietderīgus gājienus. (skatīt pielikumu Nr.2)

###### **Heuristikas izmantošana**

Heuristika novērtē stāvokļa tuvumu atrisinājumam, izmantojot iepriekš aprēķinātu tabulu. Tiek izmantoti tikai orientācijas dati (stūri un malas), lai samazinātu tabulas apjomu. Meklēšana tiek veikta tikai gadījumos, kad  $g + h \leq \text{robeža}$ , kur:

- $g$  ir soļu skaits līdz šim punktam,
- $h$  ir heuristikas vērtējums.

Heuristika garantē, ka meklēšana tiek veikta tikai iespējami perspektīvākajos zaros.

###### **Transponēšanas tabulas izmantošana**

Katrai kuba konfigurācijai tiek aprēķināts unikāls hash (piemēram, balstīts uz permutāciju/orientāciju kombināciju). Ja šis stāvoklis jau iepriekš ticis apmeklēts ar mazāku vai vienādu dziļumu, tas netiek atkārtoti apstrādāts. Šādi tiek samazināts pārmērīgs meklēšanas darbs un novērsta ciklu veidošanās.

###### **Atzarošanas stratēģijas**

Lai samazinātu iespējamo gājienu skaitu katrā solī, tiek pielietoti šādi vienkārši noteikumi:

- Neļaut atkārtot kustības pa to pašu asi (piemēram,  $R \rightarrow R2$ ).

- Izvairīties no tūlītējas kustību atsaukšanas (piemēram,  $R \rightarrow R'$ ).
- Noteikt kustību grupas, kas nedod nekādu progresu, ja tās veic secīgi.

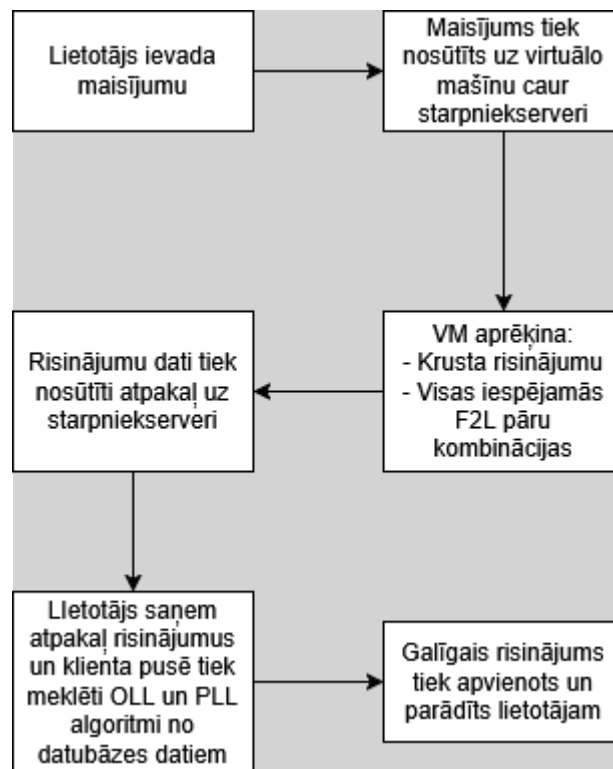
### Algoritma darbība soli pa solim

Sākuma stāvoklim tiek aprēķināta heuristikā, un noteikta sākuma robeža. Tiek sāta dziļuma meklēšana (DFS), ievērojot:

- Heuristikas robežu ( $g + h \leq \text{robeža}$ )
- Transponēšanas tabula ierobežojumus
- Atzarošanas nosacījumus

Ja risinājums netiek atrasts, robeža tiek palielināta (piem., uz nākamo minimālo  $f$  vērtību). Šis cikls atkārtojas, līdz tiek atrasts atrisinājums.

### 3.1.3. Datu plūsma



1. attēls. Datu plūsmas diagramma

#### 1. Lietotājs ievada maisījumu

Lietotājs ievada sākotnējo Rubika kuba stāvokli caur saskarni un sāk risinājumu meklēšanu.

#### 2. Maisījums tiek nosūtīts uz virtuālo mašīnu caur starpniekserveri

Sistēma nosūta šo informāciju uz virtuālo mašīnu (VM), izmantojot starpniekserveri, kas kalpo kā kontrolpunkts.

### 3. VM aprēķina:

- Krusta risinājumu (Cross)
- Visas iespējamās F2L kombinācijas

Šī daļa ir atbildīga par CFOP sākuma posmiem, kur tiek ģenerētas dažādas iespējamās F2L situācijas, balstoties uz sajaukumu.

### 4. Risinājumu dati tiek nosūtīti atpakaļ uz sistēmu

Pēc aprēķiniem, VM atgriež datus sistēmai, kas satur informāciju par krustu un F2L iespējām.

### 5. Sistēma meklē atbilstošos OLL un PLL risinājumus no datubāzes

Pamatojoties uz saņemtajām F2L situācijām, sistēma atlasa optimālākos OLL un PLL risinājumus no iepriekš sagatavotas datubāzes.

### 6. Galīgais risinājums tiek apvienots un atgriezts lietotājam

Visi posmi (Cross + F2L + OLL + PLL) tiek salikti kopā vienā secīgā risinājumā, kas tiek parādīti lietotājam kā pilns Rubika kuba atrisinājums.

#### 3.1.4. Stāvokļa reprezentācija heuristiku tabulās

- Krusts (Cross)

Krusta stāvoklis ir aprakstīts ar 4 malām: UF, UR, UL, UB.

Katrai malai attiecīgi ir iedots skaitlis, lai to vieglāk izmantot meklēšanā, kur katra mala ir numurēta no 0 līdz 3, pārējās malas, kas netiek izpētītas tiek apzīmētas ar -1, liekot tai būt jebkurai malai.

Katrai malai attiecīgi ir arī orientācija kas tiek atzīmēta kā 0 vai 1.

Kopējais stāvoklis tiek reprezentēts vienā masīvā, piemēram, (-1, 1, 2, 3, -1, -1, -1, -1, -1, -1, -1, 0, -1, 0, 0, 0, -1, -1, -1, -1, -1, -1, -1, 0)

- F2L pāri

F2L stāvoklis tiek sadalīts trijās neatkarīgās komponentēs:

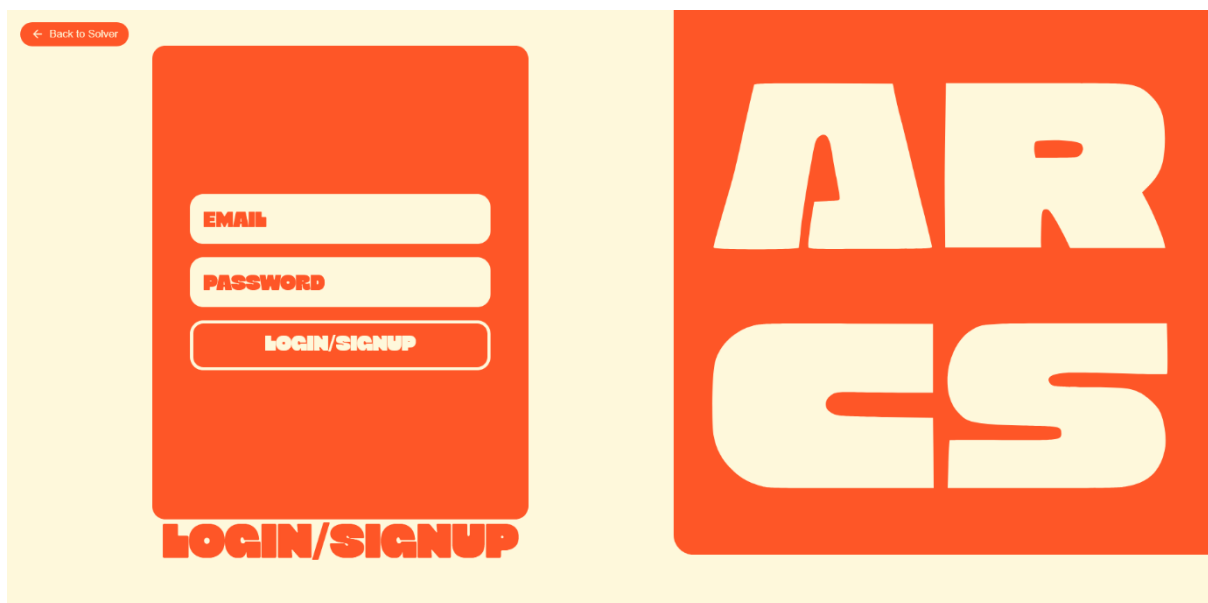
crossState — krusta malas dati (iepriekš minētajā formā),

edgeState — 4 malu pozīcijas (FR, FL, BL, BR) un orientācijas (0/1), piemēram, (-1, -1, 0, -1, 1, -1, 2, 3, -1, -1, 0, -1, 0, -1, 0, 1)

cornerState — 4 stūru pozīcijas (URF, UFL, ULB, UBR) un orientācijas (0–2), piemēram, (8, 9, -1, -1, 11, -1, -1, -1, -1, -1, 10, -1, 0, 1, -1, -1, 0, -1, -1, -1, -1, -1, 0, -1)

## 3.2. Saskarnes realizācija

### 3.2.1. Reģistrācijas/Pieslēgšanās lapa



2. attēls. Pieslēgšanās/reģistrācijas lapas saskarne

Reģistrācijas un pieslēgšanās process sākas, kad lietotājs apmeklē pieslēgšanās/reģistrācijas lapu, kurā tiek attēlots minimālistisks un vizuāli pārdomāts interfeiss. Lapas dizains balstās uz pielāgotu HTML un CSS. Formas laukos lietotājam tiek piedāvāts ievadīt savu e-pasta adresi un paroli. Abi lauki ir obligāti, un tos papildina saprotami vietturi, kas vienkāršo navigāciju.

Formas apstrāde tiek veikta ar login.js JavaScript moduli, kurš, uzspiežot "LOGIN/SIGNUP" pogu, uzsāk lietotāja autentifikācijas vai reģistrācijas procesu. Šī funkcionalitāte var izmantot ārējo autentifikācijas pakalpojumu (piemēram, Appwrite vai līdzīgu), lai pārvaldītu sesijas, izveidotu jaunu kontu vai pārbaudītu esošu lietotāju akreditācijas datus.

Lapas izkārtojums ir vienkāršs un efektīvs — galvenie elementi (forma un logotips) tiek attēloti viens otram blakām, nodrošinot fokusu un ērtu lietojamību. Autors šajā lapā izvēlējies praktisku un funkcionālu pieeju, kas atbilst lietotnes vajadzībām – ļaut lietotājiem autentificēties vai izveidot kontu ātri un droši.

### 3.2.2. Galvenā risināšanas lapa



### 3. attēls. Galvenās risināšanas lapas saskarne

Risināšanas lapa ir paredzēta Rubika kuba entuziastiem, nodrošinot ērtu un mērķtiecīgu vidi risinājumu ģenerēšanai. Interfeiss veidots ar tīru HTML, CSS un JavaScript kombināciju, kas kopā nodrošina vizuāli pārdomātu un funkcionālu lietošanas pieredzi. Lapas struktūra sadalīta divās galvenajās daļās – kreisajā pusē atrodas lietotāja ievades lauki un darbības pogas, bet labajā – vieta risinājumu attēlošanai.

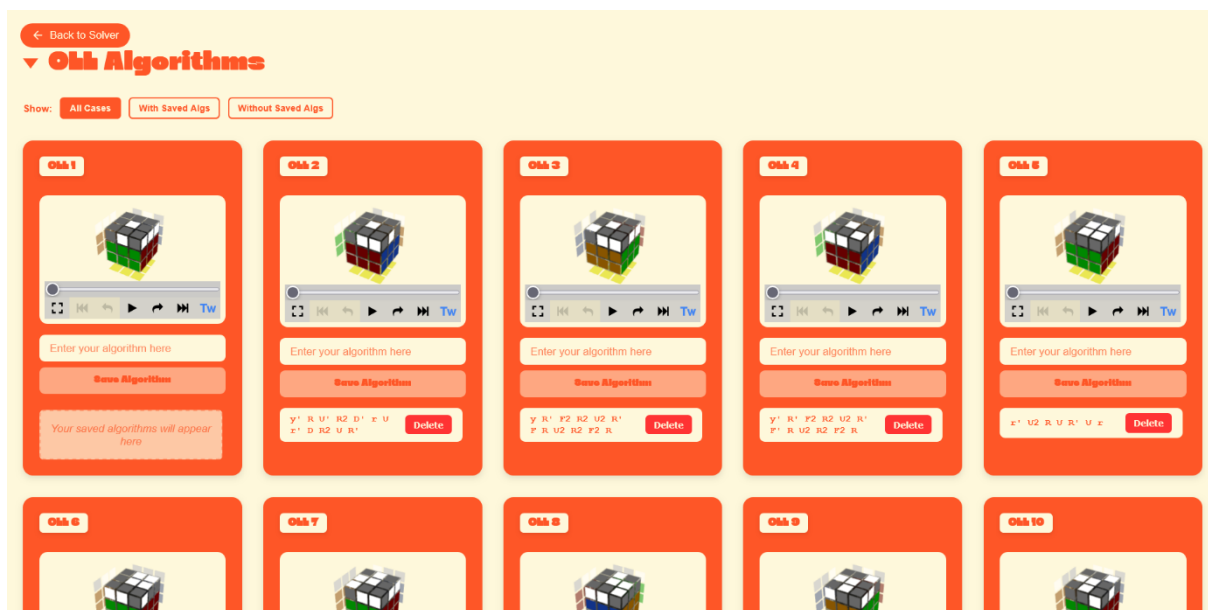
Galvenais ievades lauks ir teksta laukums, kurā lietotājs ievada sava kuba jaukšanas algoritmu. Blakus tam atrodas poga “Get Solutions”, kuru nospiežot aktivizē JavaScript moduli `getSol.js`. Šis modulis analizē ievadīto jaukumu un, izmantojot fonā definētu algoritmu, atgriež optimizētus atrisinājumus, kurus pēc tam attēlo labajā pusē — speciāli šim nolūkam paredzētā konteinerī.

Lapa arī ietver autentifikācijas pogas (login, logout, user profile), kas ir integrētas kā intuitīvas SVG ikonas augšējā joslā. Tās nodrošina sesiju pārvaldību un lietotāja profila piekļuvi, izmantojot Appwrite autentifikācijas pakalpojumus.

Lapas izkārtojums ir ne tikai estētiski pārdomāts, bet arī pilnībā responsīvs. Brīžos, kad lietotāja ierīces ekrāns kļūst šaurāks (piemēram, mobilajās ierīcēs), lapa automātiski pārkārtojas: lietotāja ievades sadaļa tiek novietota ekrāna augšpusē, bet risinājumu sadaļa – apakšā. Šāda vertikālā sadalījuma pieeja nodrošina ērtu lietojamību un optimizē vietas izmantošanu dažādos ekrāna izmēros.

Autors šeit izvēlēties pragmatisku pieeju, apvienojot tehnisko funkcionalitāti ar viegli uztveramu vizuālo izkārtojumu, kas atbilst lapas mērķim — sniegt rīku Rubika kuba atrisināšanai.

### 3.2.3. Lietotāja preferences



#### 4. attēls. Lietotāju preferenču saskarne

Šī lapa kalpo kā lietotāja Rubika kuba personīgo algoritmu krātuve, kurā iespējams pielāgot un saglabāt individuāli izvēlētos OLL un PLL risinājumus. Tā izstrādāta ar uzsvaru uz skaidrību un ērtu lietojamību, izmantojot tikai tīru HTML, CSS un JavaScript. Šāda pieeja nodrošina viegli uztveramu un stabilu vidi, kas piemērota gan datorā, gan mobilajās ierīcēs.

Lapas augšpusē atrodas atgriešanās poga uz risināšanas lapu, kas ērti ļauj pārslēgties starp algoritmu definēšanu un faktisko risināšanu. Zemāk atrodas divas galvenās funkcionālās daļas: algoritmu karšu pārskats un filtrēšanas iespējas.

Algoritmu pārskatā katram OLL un līdzīgā veidā arī PLL gadījumam paredzēta atsevišķa karte. Šīs kartes ir strukturētas tā, lai vienuviet apvienotu visu nepieciešamo informāciju konkrētā gadījuma identificēšanai un algoritma pārvaldībai. Katras kartes augšpusē redzams gadījuma nosaukums, piemēram, "PLL Aa" vai "OLL 1", un zem tā izvietots twisty-player vizualizators, kas attēlo konkrēto kuba stāvokli trīsdimensionālā skatā. Šī interaktīvā vizualizācija palīdz lietotājam uzreiz saprast konkrēto situāciju uz kuba.

Zem vizualizācijas atrodas algoritma ievades lauks ar blakus esošu "Save Algorithm" pogu. Tā sākotnēji ir deaktivizēta, un kļūst aktīva tikai tad, kad ievadīts sintaktiski pareizs un ar pareizu risinājumu algoritms. Saglabātie algoritmi tiek parādīti uzreiz zem ievades lauka, skaidrā, lasāmā formātā. Katrs no tiem ir papildināts ar dzēšanas pogu, lai ļautu lietotājam viegli pārvaldīt un mainīt savu krājumu. Šādi katra karte kļūst par mazu rediģējamu datubāzi vienam konkrētam risināšanas scenārijam.

Filtrēšanas josla, kas novietota OLL un PLL sadaļas augšdaļā, sniedz lietotājam iespēju fokusēties uz konkrētiem algoritmiem, atkarībā no to statusa. Pieejamas trīs iespējas:

- Rādīt visus gadījumus,
- Rādīt tikai tos, kuriem jau ir saglabāts vismaz viens algoritms,

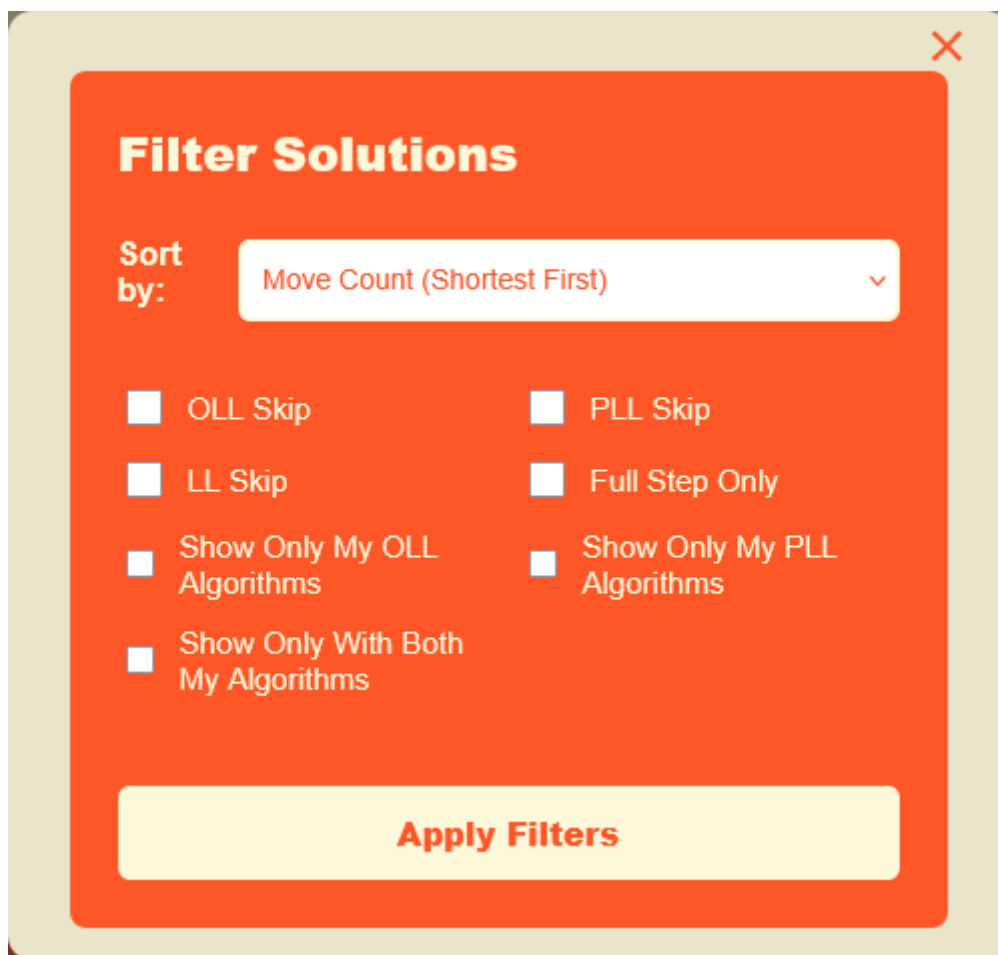
- Rādīt tikai tos gadījumus, kuri vēl nav aizpildīti.

Šī funkcionalitāte darbojas dinamiskā veidā, pateicoties JavaScript loģikai, un nodrošina ātru filtrēšanu bez lapas pārlādēšanas.

Tāpat kā pārējās projekta sastāvdaļās, arī šeit ir ievērots responsīvais dizains, kas pielāgo elementu izkārtojumu atkarībā no ekrāna izmēra. Uz mazākiem ekrāniem karšu režģis tiek pārkārtots vertikāli, nodrošinot ērtu pārskatāmību un izmantojamību arī mobilajās ierīcēs.

Šī lapa kopumā veido svarīgu Rubika kuba risinātāja sistēmas komponenti, sniedzot lietotājam pilnīgu kontroli pār saviem izvēlētajiem risinājumiem un ļaujot sistemātiski strādāt ar algoritmu krājumu. Tā kalpo gan kā praktisks darba rīks, gan kā personalizēta atmiņas datubāze, palīdzot lietotājam pilnveidot savu efektivitāti risināšanas procesā.

#### 3.2.4. Filtrēšanas modālis



5. attēls. Filtrēšanas modāļa saskarne

Risināšanas sadaļā atrodas filtrēšanas un kārtotāšanas panelis, kas ļauj lietotājam precīzi pielāgot atrasto algoritmu sarakstu atbilstoši savām vajadzībām un preferencēm. Šis panelis ir novietots skaidri redzamā vietā virs atrisinājumu rezultātiem un ļauj interaktīvi manipulēt ar parādāmajiem datiem, izmantojot JavaScript loģiku bez lapas pārlādēšanas.



Paneļa augšpusē lietotājs var izvēlēties kārtot algoritmus pēc dažādiem kritērijiem. Noklusētā izvēlne ļauj sakārtot rezultātus pēc soļu skaita augošā vai dilstošā secībā, kā arī pēc OLL vai PLL gadījuma, lai vieglāk grupētu līdzīgus scenārijus. Šī funkcionalitāte ir noderīga, ja lietotājs vēlas atrast visīsāko risinājumu vai salīdzināt dažādus algoritmus viena gadījuma ietvaros.

Zem kārtošanas izvēlnes atrodas detalizēta filtru sadaļa ar vairākiem izvēles rūtiņām. Filtri ļauj parādīt tikai tos risinājumus, kuri ietver specifiskus gadījumus: OLL Skip, PLL Skip, vai pilnībā atrisinātus pēdējos slāņus (LL Skip). Tāpat iespējams aktivizēt "Full Step Only" režīmu, lai atlasītu tikai tos risinājumus, kuros neviens solis nav izlaists. Papildu filtri ļauj ierobežot rezultātus tikai uz tiem, kuri izmanto lietotāja personīgi saglabātos OLL un/vai PLL algoritmus — vai tikai uz tiem, kuri izmanto abus vienlaicīgi. Šie personalizētie filtri palīdz uzlabot treniņu efektivitāti.

Paneļa apakšā izvietota poga "Apply Filters", kas piemēro izvēlētos filtrus un kārtošanas kritērijus meklēšanas rezultātiem. Tā darbojas nekavējoties, nodrošinot ātru un elastīgu piekļuvi optimālākajiem algoritmiem katrā konkrētajā situācijā.

### 3.3. Programmas produkta atbilstība specifikācijai

| Testpiemērs   | Testa solis | Testa soļa apraksts   | Sasniedzamais rezultāts   | Iegūtais rezultāts   | Statuss  | Piezīmes |
|---|-------------|---|---|--|----------|----------|
| Risinājumu ģenerēšana   | 1           | Ievada kuba sajaukumu, piemēram, “D' L2 F D2 R' F2 D' L' U' R2 D2 L' U F' U2 R D2 L' D' R B' L' U' R” . | Ievadītais sajaukums parādās teksta laukā.  | Sajaukums parādijās laukā.   | Izdevies |          |
|   | 2           | Nospiež pogu “Get Solutions”.   | Ievadītais sajaukums tiek uzlikts uz virtuālā kuba. Izmantojot risināšanas algoritmu lietotājam tiek atgriezti vairāki šī sajaukuma risinājumi. | Atgriezti vairāki risinājumi, kurus uzliekot uz kuba ar iepriekš definēto sajaukumu tiek izrisināts Rubika kubs. | Izdevies |          |
| Lietotāju algoritmu preferences izmantošana. Tiek izmantoti tikai lietotāja algoritmi | 1           | Pirms risinājumi tiek ģenerēti, tiek izvēlēta “Use only my algorithms” opcija.                          | Tiek iestatīts filtrs liekot sistēmai zināt, ka vajag izmantot tikai lietotāja definētos algoritmus   | Tiek iestatīts filtrs.   | Izdevies |          |
|   | 2           | Tiek nospiesta poga “Get Solutions”.  | Tiek ģenerēti risinājumi, kuru pēdējie divi soļi, OLL un PLL algoritmi tiek skatīti no lietotāja definētās tabulas,                             | Tiek atgriezti risinājumi ar zvaigznīti pie OLL un PLL soļiem.   | Izdevies |          |

|  |   |  |  |  |          |  |
|--|---|--|--|--|----------|--|
|  |   |  | un izmantoti tikai tie algoritmi, katram risinājumam pie OLL un PLL soļa vajadzētu parādīties zvaigznītei parādot lietotājam, ka tas ir viņa iepriekš definētais algoritms.  |  |          |  |
| Lietotāju algoritmu preferences izmantošana. Tiek prioritizēti lietotāja algoritmi | 1 | Pirms risinājumi tiek ģenerēti, tiek izvēlēta "Use only my algorithms" opcija. | Tiek iestatīts filtrs liekot sistēmai zināt, ka vajag prioritizēt lietotāja definētos algoritmus   | Tiek iestatīts filtrs.   | Izdevies |  |
|  | 2 | Tiek nospiesta poga "Get Solutions".   | Tiek ģenerēti risinājumi, kuru pēdējie divi soļi, OLL un PLL algoritmi tiek skatīti no lietotāja definētās tabulas un noklusējam tabulām, un katram risinājumam pie OLL un PLL soļa vajadzētu parādīties zvaigznītei parādot | Tiek atgriezti risinājumi ar zvaigznīti pie OLL un PLL soļiem un bez zvaigznītēm liecinot, ka ir gan lietotāja algoritmi, gan noklusējuma. | Izdevies |  |

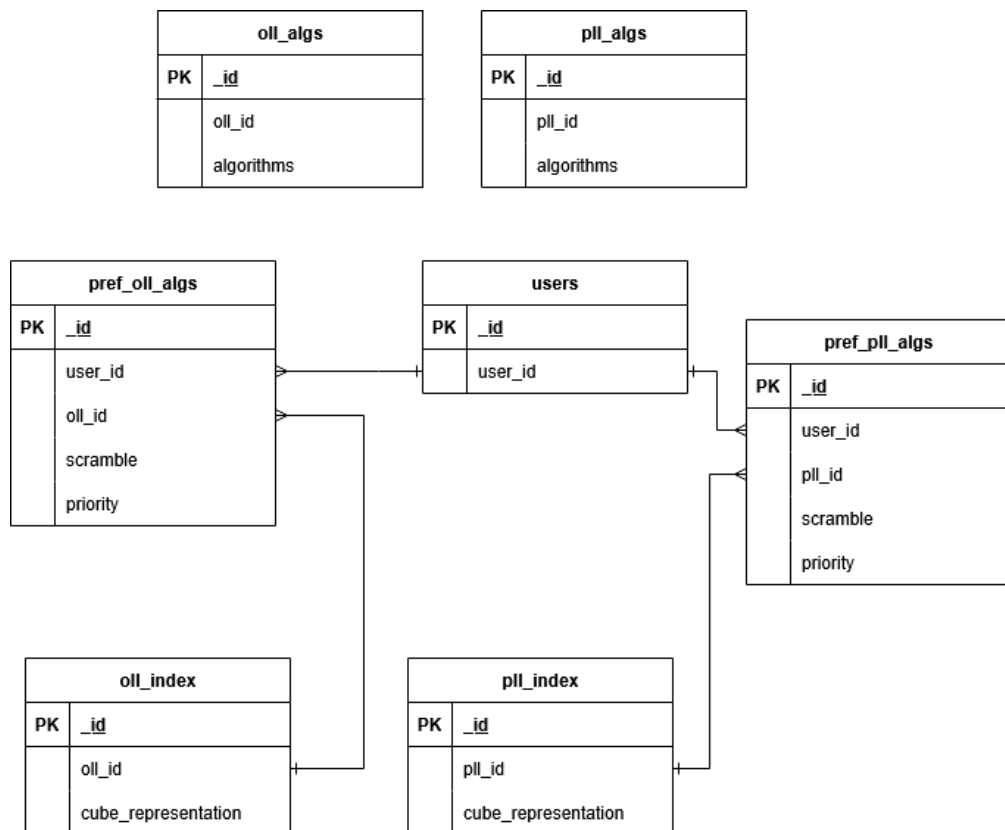
|   |   |  |   |   |          |  |
|---|---|--|---|---|----------|--|
|   |   |  | lietotājam, ka tas ir viņa iepriekš definētais algoritms.   |   |          |  |
| Lietotāju algoritmu preferences izmantošana. Tiek izmantoti noklusējuma algoritmi | 1 | Pirms risinājumi tiek ģenerēti, tiek izvēlēta "Use only my algorithms" opcija. | Tiek iestatīts filtrs lietot sistēmai zināt, ka vajag prioritizēt lietotāja definētos algoritmus.         | Tiek iestatīts filtrs.  | Izdevies |  |
|   | 2 | Tiek nospiesta poga "Get Solutions".   | Tiek ģenerēti risinājumi, kuru pēdējie divi soļi, OLL un PLL algoritmi tiek skatīti no noklusējam tabulā. | Tiek atgriezti risinājumi.  | Izdevies |  |
| Lietotāja reģistrācijas ar derīgiem akreditācijas datiem                          | 1 | E-pasta laukā tiek ievadīts derīgs e-pasts.                                    | E-pasts parādās e-pasta laukā.  | E-pasts parādās e-pasta laukā.  | Izdevies |  |
|   | 2 | Paroles laukā tiek ievadīta derīga parole.                                     | Parole parādās ar paslēptām rakstzīmēm paroles laukā.   | Parole parādās ar paslēptām rakstzīmēm paroles laukā.   | Izdevies |  |
|   | 3 | Tiek nospiesta poga "LOGIN/SIGNUP"   | Tiek validēti dati un tiek izveidots lietotāja kods.  | Jauns lietotāja kods tiek izveidots, nekādi paziņojumi neizleca ārā un lietotājs tiek pāradresēts uz galveno lapu | Izdevies |  |
| Lietotāja reģistrācijas ar  | 1 | E-pasta laukā tiek ievadīts nederīgs e-pasts, piemēram                         | E-pasts parādās e-pasta laukā.  | E-pasts parādās e-pasta laukā.  | Izdevies |  |

|   |   |   |   |   |            |  |
|---|---|---|---|---|------------|--|
| nederīgu e-pasta adresi   |   | e-pasta adrese, kas nav pēc standartiem kā "email@email.com"                |   |   |            |  |
|   | 2 | Paroles laukā tiek ievadīta derīga parole.                                  | Parole parādās ar paslēptām rakstzīmēm paroles laukā. | Parole parādās ar paslēptām rakstzīmēm paroles laukā.                       | Izdevies   |  |
|   | 3 | Tiek nospiesta poga "LOGIN/SIGNUP"  | Tiek validēti dati un tiek izveidots lietotāja konts. | Jauns lietotāja konts netiek izveidots, jo e-pasta adrese nav pareiza.      | Neizdevies |  |
| Lietotāja reģistrācijas ar paroli kuras rakstzīmju daudzums ir mazāks par 6 | 1 | E-pasta laukā tiek ievadīts derīgs e-pasts.                                 | E-pasts parādās e-pasta laukā.                        | E-pasts parādās e-pasta laukā.  | Izdevies   |  |
|   | 2 | Paroles laukā tiek ievadīta nederīga parole, kura ir zem 6 rakstzīmēm gara. | Parole parādās ar paslēptām rakstzīmēm paroles laukā. | Parole parādās ar paslēptām rakstzīmēm paroles laukā.                       | Izdevies   |  |
|   | 3 | Tiek nospiesta poga "LOGIN/SIGNUP"  | Tiek validēti dati un tiek izveidots lietotāja konts. | Jauns lietotāja konts netiek izveidots, jo parole ir īsāka par 6 rakstzīmēm | Neizdevies |  |
| Lietotājs pieslēdzas ar derīgiem akreditācijas datiem                       | 1 | E-pasta laukā tiek ievadīts derīgs e-pasts.                                 | E-pasts parādās e-pasta laukā.                        | E-pasts parādās e-pasta laukā.  | Izdevies   |  |
|   | 2 | Paroles laukā tiek ievadīta derīga parole.                                  | Parole parādās ar paslēptām rakstzīmēm paroles laukā. | Parole parādās ar paslēptām rakstzīmēm paroles laukā.                       | Izdevies   |  |
|   | 3 | Tiek nospiesta poga "LOGIN/SIGNUP"  | Tiek validēti dati un lietotājs pieslēdzas sistēmai   | Lietotājs tiek pāradresēts uz galveno lapu, jo ir                           | Izdevies   |  |

|   |   |   |  |   |                 |  |
|---|---|---|--|---|-----------------|--|
|   |   |   |  | veiksmīgi<br>pieslēdzies<br>izmantojot savu<br>iepriekš reģistrēto<br>kontu.  |                 |  |
| Lietotāja<br>pieslēgšanās ar<br>nereģistrētu e-<br>pastu.                           | 1 | E-pasta laukā tiek ievadīts<br>nereģistrēts e-pasts.  | E-pasts parādās e-<br>pasta laukā.                             | E-pasts parādās e-<br>pasta laukā.  | Izdevies        |  |
|   | 2 | Paroles laukā tiek ievadīta<br>nederīga parole, kura ir zem<br>6 rakstzīmēm gara.   | Parole parādās ar<br>paslēptām<br>rakstzīmēm paroles<br>laukā. | Parole parādās ar<br>paslēptām<br>rakstzīmēm<br>paroles laukā.  | Izdevies        |  |
|   | 3 | Tiek nospiesta poga<br>“LOGIN/SIGNUP”   | Tiek validēti dati<br>un lietotājs tiek<br>pieslēgts sistēmai. | Lietotājs netiek<br>pievienots pie<br>sistēmas, bet tam<br>uzreiz mēģina<br>izveidot kontu, un<br>ja ievadītā e-pasta<br>adrese neeksistē<br>sistēma lietotājs<br>tiek piereģistrēts<br>un pieslēgts<br>sistēmai. | Daļēji izdevies |  |
| Lietotāja<br>pieslēgšanās ar<br>paroli, kura nav<br>derīga attiecīgajam<br>e-pastam | 1 | E-pasta laukā tiek ievadīts<br>derīgs e-pasts.  | E-pasts parādās e-<br>pasta laukā.                             | E-pasts parādās e-<br>pasta laukā.  | Izdevies        |  |
|   | 2 | Paroles laukā tiek ievadīta<br>nederīga parole, kura ir<br>garāka par 6 rakstzīmēm, bet<br>nav pareiza priekš attiecīgā<br>e-pasta. | Parole parādās ar<br>paslēptām<br>rakstzīmēm paroles<br>laukā. | Parole parādās ar<br>paslēptām<br>rakstzīmēm<br>paroles laukā.  | Izdevies        |  |
|   | 3 | Tiek nospiesta poga<br>“LOGIN/SIGNUP”.  | Tiek validēti dati<br>un lietotājs tiek<br>pieslēgts sistēmai. | Lietotājs netiek<br>pievienots pie<br>sistēmas un tiek  | Neizdevies      |  |

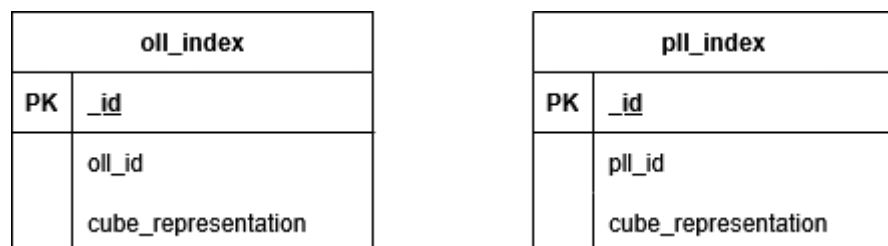
|  |   |  |   |  |            |  |
|--|---|--|---|--|------------|--|
|  |   |  |   | izsviests paziņojums, ka ievadītā parole nav derīga.                                   |            |  |
| Lietotāja OLL/PLL preferences definēšana ar derīgu algoritmu   | 1 | Tiek ievadīts algoritms, kas ir derīgs risinājums izvēlētajam OLL/PLL gadījumam.   | Ievadot algoritmu tas automātiski tiek validēts uz virtuālā Rubika kuba.  | Ievadītais algoritms tiek validēts un poga "Save Algorithm" tiek atļauta nospiešanai   | Izdevies   |  |
|  | 2 | Nospiesta poga "Save Algorithm"  | Ievadītais algoritms tiek vēlreiz validēts un tiek pievienots datu bāzei. | Algoritms veiksmīgi pievienots datu bāzei un algoritmu sarakstam.                      | Izdevies   |  |
| Lietotāja OLL/PLL preferences definēšana ar nederīgu algoritmu | 1 | Tiek ievadīts algoritms, kas ir nederīgs risinājums izvēlētajam OLL/PLL gadījumam. | Ievadot algoritmu tas automātiski tiek validēts uz virtuālā Rubika kuba.  | Ievadītais algoritms tiek validēts un poga "Save Algorithm" netiek atļauta nospiešanai | Neizdevies |  |
|  | 2 | Nospiesta poga "Save Algorithm"  | Ievadītais algoritms tiek vēlreiz validēts un tiek pievienots datu bāzei. | Algoritms netiek saglabāts, jo pati poga nekad netika nospiesta.                       | Neizdevies |  |

## 4. PROGRAMMAS PRODUKTA DATU STRUKTŪRAS APRAKSTS



6. attēls. ER diagramma

### 4.1. Index tabulas



7. attēls. Index tabulas

Šīs tabulas ir atbildīgas par katra OLL un PLL gadījuma reprezentācijas saglabāšanu attiecīgajam OLL vai PLL gadījumam.

#### 1. oll\_index tabula

- **\_id:** identifikators, auto ģenerēts.
- **oll\_id:** int, OLL gadījuma identifikators
- **cube\_representation:** 3D int masīvs, kur tiek reprezentēts, kā izskatās šis oll\_id uz virtuālā Rubika kuba



## 2. pll\_index tabula

- **\_id**: identifikators, auto ģenerēts.
- **pll\_id**: string, PLL gadījuma identifikators
- **cube\_representation**: 3D masīvs, kur tiek reprezentēts, kā izskatās šis pll\_id uz virtuālā Rubika kuba

## 4.2. Algoritmu tabulas

| oll_algs |            | pll_algs |            |
|----------|------------|----------|------------|
| PK       | <u>_id</u> | PK       | <u>_id</u> |
|          | oll_id     |          | pll_id     |
|          | algorithms |          | algorithms |

8. attēls. Algoritmu tabulas

Šīs tabulas satur iepriekš atlasītus algoritmus priekš katra OLL un PLL gadījuma, kurus izmanto meklēšanas laikā.

## 3. oll\_algs tabula

- **\_id**: identifikators, auto ģenerēts.
- **oll\_id**: int, OLL gadījuma identifikators
- **algorithms**: string masīvs, kur tiek saglabāti vairāki algoritmi katram OLL gadījumam

## 4. pll\_algs tabula

- **\_id**: identifikators, auto ģenerēts.
- **pll\_id**: string, OLL gadījuma identifikators
- **algorithms**: string masīvs, kur tiek saglabāti vairāki algoritmi katram PLL gadījumam

## 4.3. Lietotāji

| users |            |
|-------|------------|
| PK    | <u>_id</u> |
|       | user_id    |

9. attēls. Lietotāju tabula

Šeit tiek saglabāti lietotāja id no Appwrite iegūti.

## 5. users tabula

- **\_id**: identifikators, auto ģenerēts.
- **user\_id**: string, unikāls identifikators no Appwrite

#### 4.4. Lietotāja Algoritmi

| pref_oll_algs |            | pref_pll_algs |            |
|---------------|------------|---------------|------------|
| PK            | <u>_id</u> | PK            | <u>_id</u> |
|               | user_id    |               | user_id    |
|               | oll_id     |               | pll_id     |
|               | scramble   |               | scramble   |
|               | priority   |               | priority   |

10. attēls. Lietotāju preferenču tabulas

Šajās tabulās tiek saglabāti lietotāja OLL un PLL algoritmi, uz kuriem risinātājs paļausies, ja lietotājs izvēlēties to opciju.

6. preff\_oll\_algs tabula

- **\_id:** identifikators, auto ģenerēts.
- **user\_id:** string, kas ir iegūts no **users** tabulas
- **oll\_id:** int, kas ir iegūts no **oll\_index** tabulas
- **scramble:** string, kas satur lietotāja preferenci
- **priority:** int, kārtas numurs, kā parādīties sarakstā

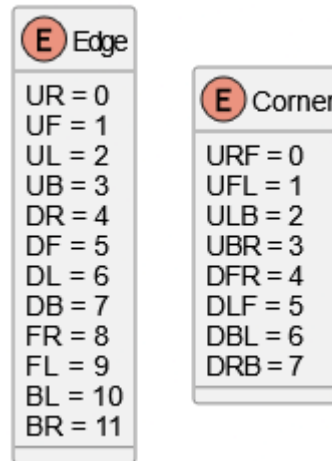
7. preff\_pll\_algs tabula

- **\_id:** identifikators, auto ģenerēts.
- **user\_id:** string, kas ir iegūts no **users** tabulas
- **pll\_id:** string, kas ir iegūts no **pll\_index** tabulas
- **scramble:** string, kas satur lietotāja preferenci
- **priority:** int, kārtas numurs, kā parādīties sarakstā

#### 4.5. Klašu diagramma

Šī klašu diagramma attēlo Rubika kuba risināšanas sistēmu, kurā CubieCube klase pārstāv kuba stāvokli kā stūrīšu un malu pozīcijas un orientācijas. Risināšanai tiek izmantots IDA\* algoritms, kas ir ieviests IDA\_star\_cross klasē priekš krusta un IDA\_star\_F2L klasē priekš F2L posma, izmantojot heuristikas vērtēšanu. Heuristikas dati tiek ielādēti caur Tableloader klasi, lai paātrinātu meklēšanu un samazinātu skaitļošanas laiku. Klases Edge un Corner definē kuba malu un stūrīšu numerāciju, ko izmanto pozīciju aprēķinos. (skatīt pielikumu Nr.3)

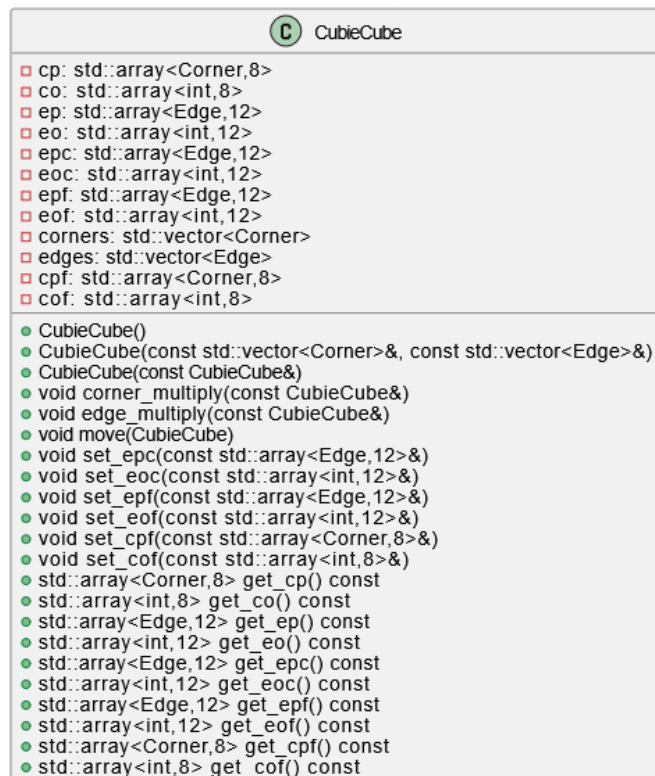
## 4.6. ENUM Edge un Corner klases



11. attēls. Klašu diagrammas enum klasēm

Edge un Corner ir enumerācijas (enum klases), kas definē visus Rubika kuba malu (Edge) un stūrišu (Corner) gabalus pēc to sākotnējās pozīcijas. Šie enumi ļauj vienkārši un precīzi atsaukties uz konkrētiem kuba elementiem, piemēram, UR (augšējā-labā mala) vai URF (augšējā-labā-priekšējā stūris), un tiek izmantoti iekšēji visās klasēs, kur tiek manipulēts ar kuba stāvokli. Tie padara kodu saprotamāku un novērš kļūdas, kas varētu rasties, lietojot tikai skaitļus.

## 4.7. CubieCube klase



12. attēls. CubieCube klases diagramma

CubieCube ir klase, kas pārstāv Rubika kuba stāvokli, balstoties uz gabaliņu (cubie) pozīcijām un orientācijām. Tā vietā, lai skatītos uz krāsu izvietojumu uz sejas, tā glabā konkrēti, kur katrs stūris un mala atrodas un kā tie ir pagriezti. Šī pieeja ļauj strādāt daudz efektīvāk algoritmiski, jo viss ir skaitliski un viegli kombinējams.

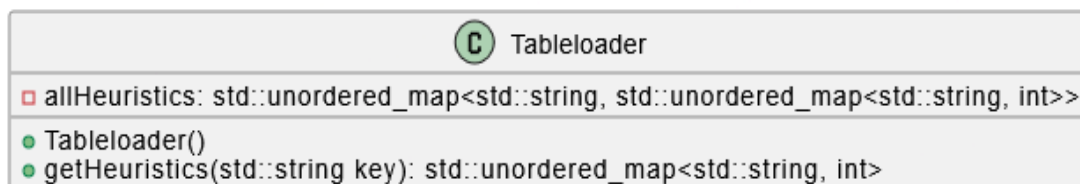
Galvenie lauki:

- cp, co: stūru pozīcijas un orientācijas (kopā 8 gabali).
- ep, eo: malu pozīcijas un orientācijas (kopā 12 gabali).
- papildus ir epc, eoc, cpf, cof, u.c., kas tiek izmantoti optimizācijai dažādos posmos, piemēram, F2L vai Cross, kur nevajag visu kuba informāciju.

Galvenās funkcijas ļauj piemērot kustības (move()), kombinēt vairākus kuba stāvokļus (corner\_multiply(), edge\_multiply()), kā arī iegūt vai iestatīt konkrētas pozīcijas (get\_\*, set\_\*). Tas viss padara šo klasi par centrālo komponenti – visa risināšana notiek, manipulējot ar CubieCube objektiem.

Šo klasi izmanto visi algoritmi, jo bez tās nav iespējams normāli izsekot, kurā stāvoklī kubs atrodas. Tā ir pamatvienība visā sistēmā.

## 4.8. Tableloader klase



### 13. attēls. Tableloader klases diagramma

Tableloader ir klase, kas atbild par iepriekš aprēķināto heuristika tabulu ielādi no failiem. Šīs tabulas satur minimālo kustību skaitu no konkrēta kuba stāvokļa līdz atrisinātam stāvoklim, un tās ir būtiskas IDA\* algoritma efektīvai darbībai. Bez tām algoritms būtu pārāk lēns, jo katru stāvokli vajadzētu novērtēt "no nulles".

Klase tiek izmantota, lai no failiem nolasītu šos datus (parasti binārus vai saspiestus formātos), atšifrētu tos un noglabātu atmiņā kā masīvus, kurus vēlāk izmanto IDA\* algoritms. Parasti tās metodes ir statiskas, un tā strādā tikai kā datu ielādētājs – neko neaprēķina, tikai padod gatavu heuristikas vērtību attiecīgajam stāvoklim. Tātad – šī klase nodrošina ātru piekļuvi svarīgākajai informācijai risināšanas laikā.

## 4.9. IDA\_Star\_F2L un IDA\_Star\_Cross klases



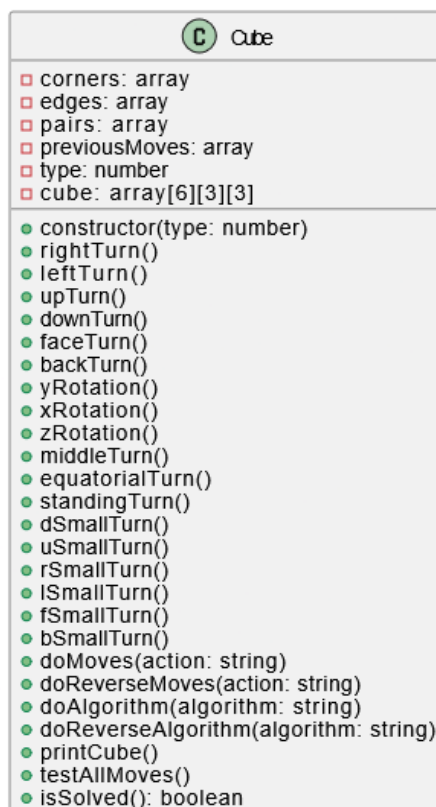
### 14. attēls. IDA\* algoritmu klases diagrammas

IDA\_star\_cross ir klase, kas izmanto IDA\* algoritmu, lai atrisinātu krusta posmu Rubika kubā. Tā meklē minimālu kustību virkni, kas noved malu gabalus uz pareizajām vietām uz apakšējās sejas, saglabājot pareizu orientāciju. Šajā posmā galvenokārt tiek ņemti vērā tikai daži kuba elementi (parasti 4 malas), tāpēc stāvokļa reprezentācija un heuristika tiek vienkāršota – izmanto tikai daļu no CubieCube datiem. Algoritms rekursīvi ģenerē kustību secības un, izmantojot heuristiku no Tableloader, pārtrauc meklēšanu, kad ir atrasta optimāla vai pietiekami laba secība.

IDA\_star\_F2L risina nākamo posmu – F2L, kur jāievieto 4 stūra-malas pāri pareizajās vietās. Šis posms ir daudz sarežģītāks, jo tajā jāņem vērā gan stūri, gan malas, to savstarpējās attiecības, orientācijas un pozīcijas. IDA\_star\_F2L izmanto līdzīgu IDA\* pieeju kā krustam, bet ar krietni sarežģītāku stāvokļa vērtējumu un dziļāku meklēšanu. Arī šeit tiek izmantotas iepriekš ielādētās heuristikas tabulas no Tableloader, kas būtiski samazina meklēšanas dziļumu. Kustības tiek ģenerētas ar noteiktiem ierobežojumiem, lai izvairītos no bezjēdzīgas atgriešanās vai simetrisku kombināciju pārbaudes.

Abas klases izmanto CubieCube, lai aprakstītu un modificētu kuba stāvokli, un bez iepriekš ielādētajām tabulām no Tableloader šie algoritmi nebūtu praktiski izmantojami – tie kļūtu pārāk lēni.

#### 4.10. RubikCube klase



15. attēls. JavaScript Cube klases diagramma

Cube ir klienta pusē izmantota klase, kas reprezentē Rubika kubu virtuāli. Tā kalpo kā neatkarīgs modelis kuba manipulācijai, simulācijai un analīzei pārlūkprogrammā. Klase ir konstruēta kā pilna kuba datu un darbību ietvars, ko iespējams izmantot testēšanai, kustību vizualizēšanai vai algoritmu eksperimentiem bez reāla kuba vai servera apstrādes.

Šī klase satur iekšējo reprezentāciju ar sešām sejām (`cube: array[6][3][3]`) un arī strukturētiem masīviem malām, stūriem un F2L pāriem (`edges`, `corners`, `pairs`), kas ļauj izveidot uzlabotu stāvokļa izsekošanu un pāreju izpildi. `previousMoves` tiek izmantots, lai saglabātu kustību vēsturi – tas noder algoritmu atkātošanai.

Kustību metodes (`rightTurn()`, `upTurn()` u.c.) ļauj precīzi manipulēt kuba stāvokli, ieskaitot arī papildu rotācijas (`xRotation()`, `middleTurn()` u.c.) un “wide” WCA stila pagriezienus (`rSmallTurn()` u.c.). Ir nodrošinātas arī metodes algoritmu virknēm (`doMoves()`, `doAlgorithm()`) un to inversajiem gājieniem (`doReverseMoves()`, `doReverseAlgorithm()`), kas ļauj ērti eksperimentēt ar jebkuru formālu gājienu virkni.

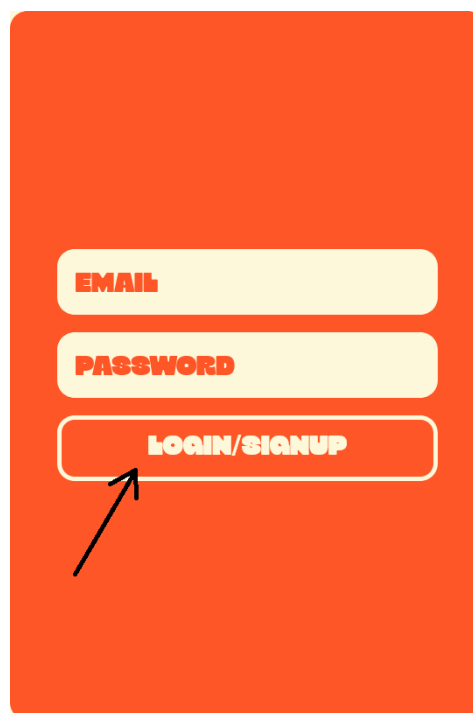
Lai analizētu rezultātu vai testētu sistēmas pareizību, `printCube()` izvada kuba stāvokli, bet `testAllMoves()` ļauj pārbaudīt, vai visas kustības funkcionē korekti. `isSolved()` sniedz vienkāršu boolean pārbaudi par to, vai kubs ir pilnībā atrisināts.

Šī klase ir būtiska visām klienta puses darbībām ar Rubika kubu – no algoritmu testēšanas līdz vizuālai reprezentācijai – un ir pamats citu algoritmisko klašu, kā `IDA_star_cross` un `IDA_star_F2L`, darbībai. Tā nodrošina manipulācijas API, uz kura balstās viss pārējais kuba loģikas slānis.

## 5. PROGRAMMAS PRODUKTA LIETOTĀJA CEĻVEDIS

### 5.1. Reģistrācija/ Pierakstīšanās

- Atveriet reģistrācijas/pieslēgšanās lapu.
- Ievadiet derīgus akreditācijas datus:
  - E-pastu: Ja vēlaties reģistrēties, ievadiet unikālu e-pasta adresi kas vēl nav izmantots konta izveidošanai sistēmā. Ja vēlaties pieslēgties ievadiet derīgu e-pasta adresi kas ir sistēmā.
  - Paroli: Priekš reģistrācijas ievadiet paroli kas ir garāka par 6 rakstzīmēm, bet ja vēlaties pieslēgties ievadiet priekš e-pasta attiecīgu paroli lai piekļūtu pie sistēmas.
- Ievadot visus akreditācijas datus attiecīgajos laukos, nospiediet pogu “LOGIN/SIGNUP”

A screenshot of a registration and login form. It features three input fields stacked vertically on an orange background. The first field is labeled 'EMAIL', the second 'PASSWORD', and the third 'LOGIN/SIGNUP'. A black arrow points to the 'LOGIN/SIGNUP' button.

16. attēls. Pieslēgšanās/Reģistrācijas konteineris

- Ja pieslēgšanās/reģistrācija ir bijusi veiksmīga, jūs tiksiet pāradresēts uz galveno lapu.

### 5.2. Risinājumu meklēšana

- Atveriet galveno lapu.
- Ievadiet vēlamo maisījumu attiecīgajā ievades laukā:

**ENTER SCRAMBLE HERE**

17. attēls. Maisījuma lauks



- Kad ir ievadīts maisījums, lai meklētu risinājumus nospiediet pogu “Get Solutions”



18. attēls. **Risinātāja sākšanas poga**

- Nospiežot pogu, meklēšana uzreiz sāksies. Meklēšana aizņem kādu laiku, tāpēc ir iespēja apskatīt pašreizējos risinājumus un meklēšanas gaitā var sekot līdzi kādi risinājumi ir atrasti nospiežot pogu “Show Current Solutions”:



19. attēls. **Progresu konteīnera sekcija**

- Tiks parādīti kādi risinājumi ir pašlaik atrasti un nospiežot uz pogas “View 3D”, varēs apskatīt izvēlēto risinājumu detalizētāk:

**Solution 1**


42 moves

**CROSS:** U B R2 U L2 D2 B2

**F2L:**  
PAIR 1: F B' U' B F'  
PAIR 2: R D B D' U R'  
PAIR 3: U2 B' U2 B  
PAIR 4: R U R' U2 R U R'

**OLL:** y2 f U R U' R' f' // *OLL 45*

**PLL:** U M2 U' M' U2 M U' M2 // *PLL Ub*



**View 3D**

20. attēls. Risinājuma kartiņa

### 5.3. Risinājumu filtrēšana

- Sekojot “Risinājuma meklēšanas” soļiem un tiekot līdz punktam, kur ir redzamas visas kartītes, nospiediet pogu “Filter solutions”:

# SOLUTIONS

Solutions found: 7797

Search progress: 317/432

 Filter Solutions

 Refresh Solutions

Cancel Solve

**Solution 1**

42 moves

**CROSS:** U B R2 U L2 D2 B2

**F2L:**  
PAIR 1: F B' U' B F'  
PAIR 2: R D B D' U R'  
PAIR 3: U2 B' U2 B  
PAIR 4: R U R' U2 R U R'

**OLL:** y2 f U R U' R' f // **OLL 45**

**PLL:** U M2 U' M' U2 M U' M2 // **PLL Ub**

View 3D

**Solution 2**

43 moves

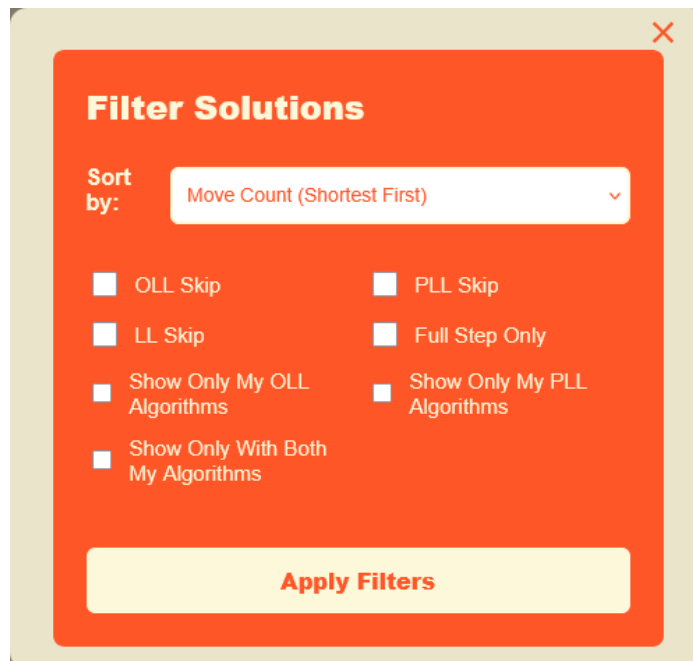
**CROSS:** U B R2 U L2 D2 B2

**F2L:**  
PAIR 1: F B' U' B F'

21. attēls. **Risinājumu sekcija**

- Nospiežot pogu atvērsies modāls, kur būs redzami visi filtri:

43



22. attēls. Filtru modālis

- Šeit variet izvēlēties starp vairākiem filtriem, kas samazinās risinājumu daudzumu precizējot vēlamu iznākumu  
Atceries – “Show Only My PLL Algorithms”, “Show Only My PLL Algorithms” un “Show Only With Both My Algorithms” opcijas parādās tikai ja esat pieslēdzies kā lietotājs sistēmai un esat pievienojuši preferences OLL vai PLL gadījumiem.
- Izvēloties filtrus nospiediet pogu “Apply Filters”, lai piemērotu filtrus rezultātiem.
- Tagad risinājumu sadaļā parādīsies risinājumi ar attiecīgajiem filtriem.

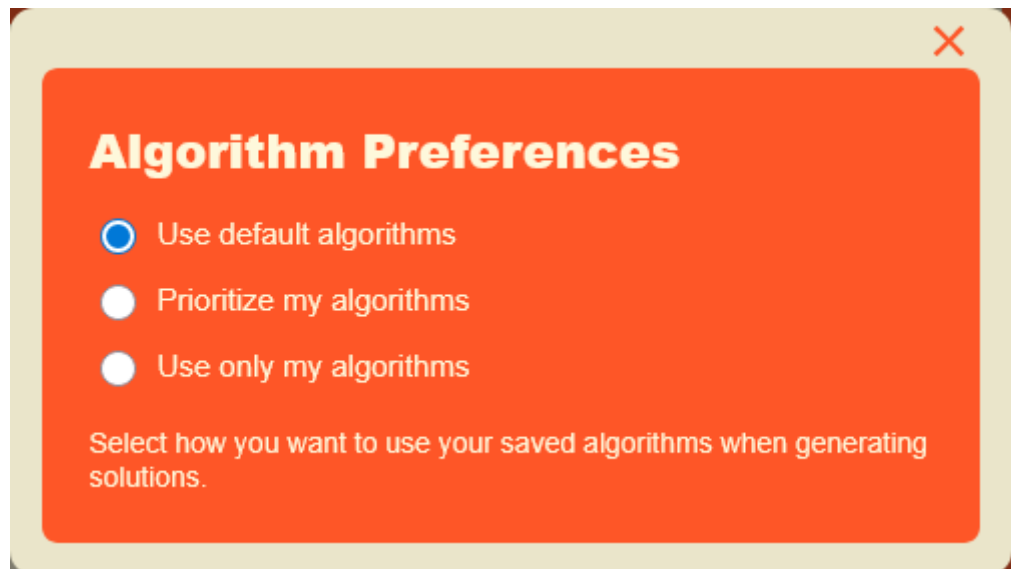
#### 5.4. Algoritmu preferenču filtrēšana

- Pirms risinājuma sākšanas var pievienot algoritmu filtrus, kas noteiks vai izmantot noklusētos OLL/PLL algoritmus, prioritizēt lietotāja algoritmus vai izmantot tikai lietotāja algoritmus.
- Atceries šos filtrus var izmantot tikai, ja esat pieslēdzies pie sistēmas un esat definējuši vizmas vienu preferenci lietotāju sadaļā.
- Nospiediet pogu “Algorithm preferences”:



23. attēls. Algoritmu preferenču izvēles poga

- Atvērsies modālis, kur izvēlaties vēlamu filtru:



24. attēls. Algoritmu preferenču modālis

- Aizveriet modāli ar “X” labajā augšējā stūrī un sāciet risinājumu meklēšanu. Atrastie risinājumi būs pēc noteiktā filtra.

### 5.5. Algoritmu preferenču pievienošana

- Lai pievienotu preferenci noteiktam OLL vai PLL gadījuma pārējiet uz lietotāja lapu.
- Atceraties, ka preferences var pievienot tikai lietotāji, kuri ir pieslēgušies pie sistēmas.
- Izvēlies vienu no OLL vai PLL gadījumiem, kam vēlaties pievienot preferencei un ievadiet attiecīgu algoritmu priekš tā gadījuma.
- Ja algoritms ir derīgs un tas tiešām atbilst OLL vai PLL gadījumam, tad poga “Save Algorithm” būs pieejama un to būs iespējams nospiegt.

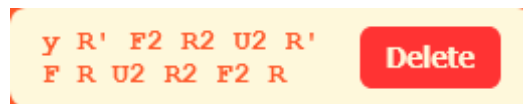


24. attēls. Saglabāt algoritmu pogas validācija

- Nospiežot pogu, tiek veikta validācija un ja ierakstītais algoritms ir derīgs tad tas parādīsies algoritmu sadaļā un algoritms tiks saglabāts datubāzē.

### 5.6. Algoritmu preferenču dzēšana

- Attiecīgi kā ar pievienošanu, algoritmus arī var dzēst, lai to izdarītu vajag būt vizmas vienai algoritma preferencei saglabātai.
- Lai izdzēstu nospiediet pogu “Delete”, blakus attiecīgajam algoritmam ko vēlaties izdzēst:



25. attēls. Saglabātais algoritms

- Nospiežot pogu, algoritms tiks izdzēst gan no datubāzes, gan no saraksta.

## SECINĀJUMI

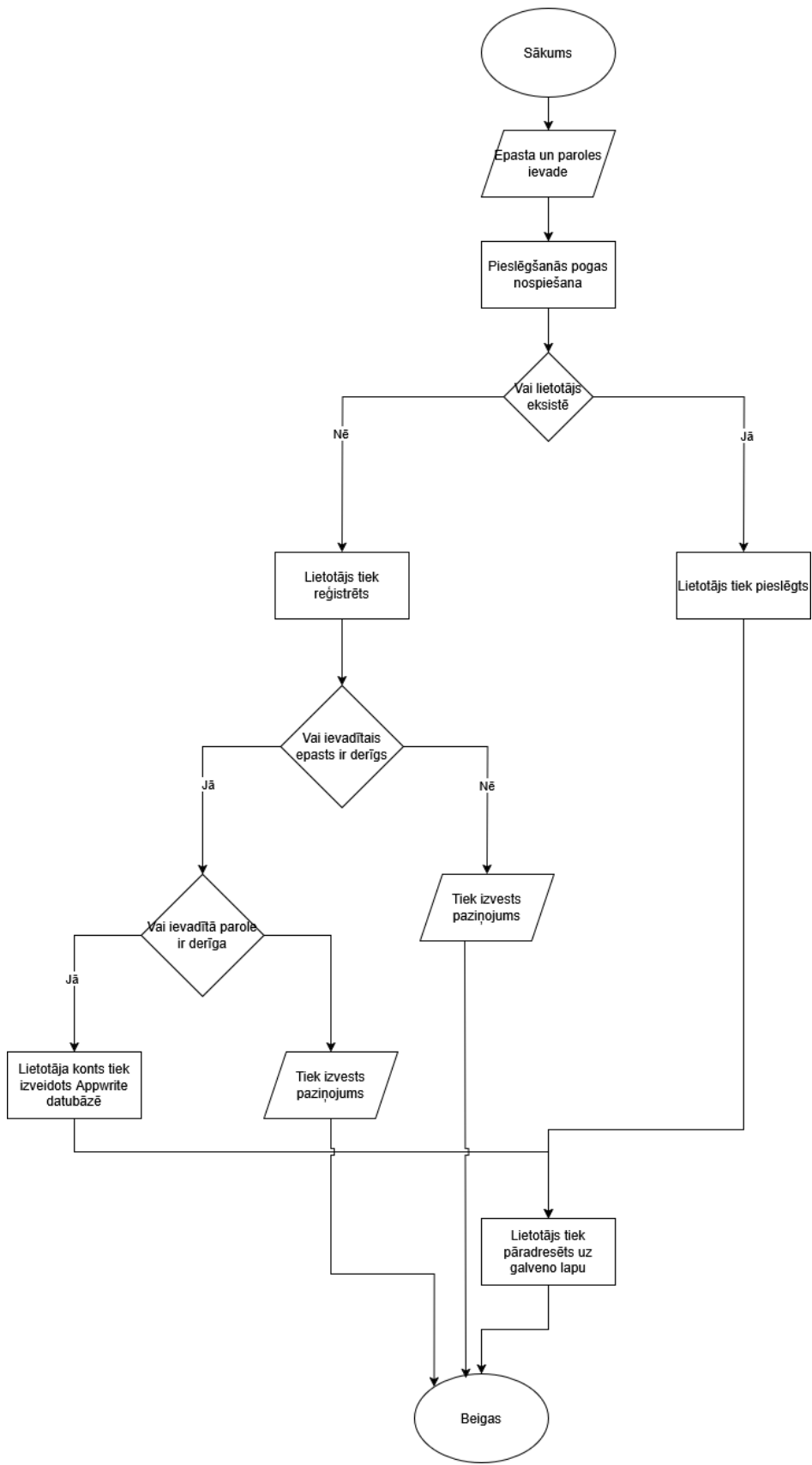
1. Funkcionālās prasības un nefunkcionālās prasības analīze palīdz precīzāk izprast sistēmas darbību un nepieciešamās konfigurācijas sistēmas uzturēšanai.
2. Veiktspējas, drošības, lietojamības un elastības kritēriju noteikšana palīdz nodrošināt, ka sistēma darbojas optimāli, atbilst lietotāju prasībām un ir droša. Šie kritēriji arī veicina elastību, ļaujot sistēmai pielāgoties dažādiem lietotāju scenārijiem un saglabājot augstu veiktspēju.
3. Sistēmas arhitektūras izstrāde nodrošina skaidru un efektīvu struktūru, kas atvieglo gan sistēmas izstrādi, gan uzturēšanu. Šāda arhitektūra ļauj viegli integrēt jaunas funkcionalitātes un nodrošina labu sadalījumu starp dažādiem sistēmas slāņiem, tādējādi uzlabojot tās mērogojamību.
4. Projektētais datubāzes risinājums atbilst prasībām, efektīvi glabājot OLL un PLL soļus, nodrošinot ātru piekļuvi un datu integritāti. Tas ļauj ērti apstrādāt lielus datu apjomus, kas ir būtiski optimizēta Rubika kuba risināšanai.
5. Sasaistes izveide starp IDA\* algoritmu un datubāzi nodrošina precīzu datu apstrādi un paātrina meklēšanas procesu, uzlabojot sistēmas kopējo efektivitāti. Šāda sasaiste arī ļauj algoritmam izmantot optimizētos datus, samazinot aprēķinu laiku.
6. IDA\* algoritma implementēšana un optimizācija palīdz uzlabot sistēmas veiktspēju un uzturēšanu, padarot kodu vieglāk saprotamu un optimizējot algoritma darbību. Tas nodrošina ātrāku un efektīvāku risinājumu Rubika kuba meklēšanā.
7. Lietotāja ceļveža izstrāde palīdz lietotājiem ātri apgūt sistēmas pamatfunkcionalitāti, nodrošinot skaidru un saprotamu pamācību. Tas veicina lietotāja pieredzi un samazina mācīšanās laiku, ļaujot lietotājiem ātrāk sasniegt vēlamu rezultātu.
8. Rubika kuba risināšanas sistēma pilnībā īsteno darba mērķi: CFOP metodes optimizācija ar IDA\* un datubāžu sasaisti. Lietotāji var personalizēt OLL/PLL algoritmus un filtrēt risinājumus.

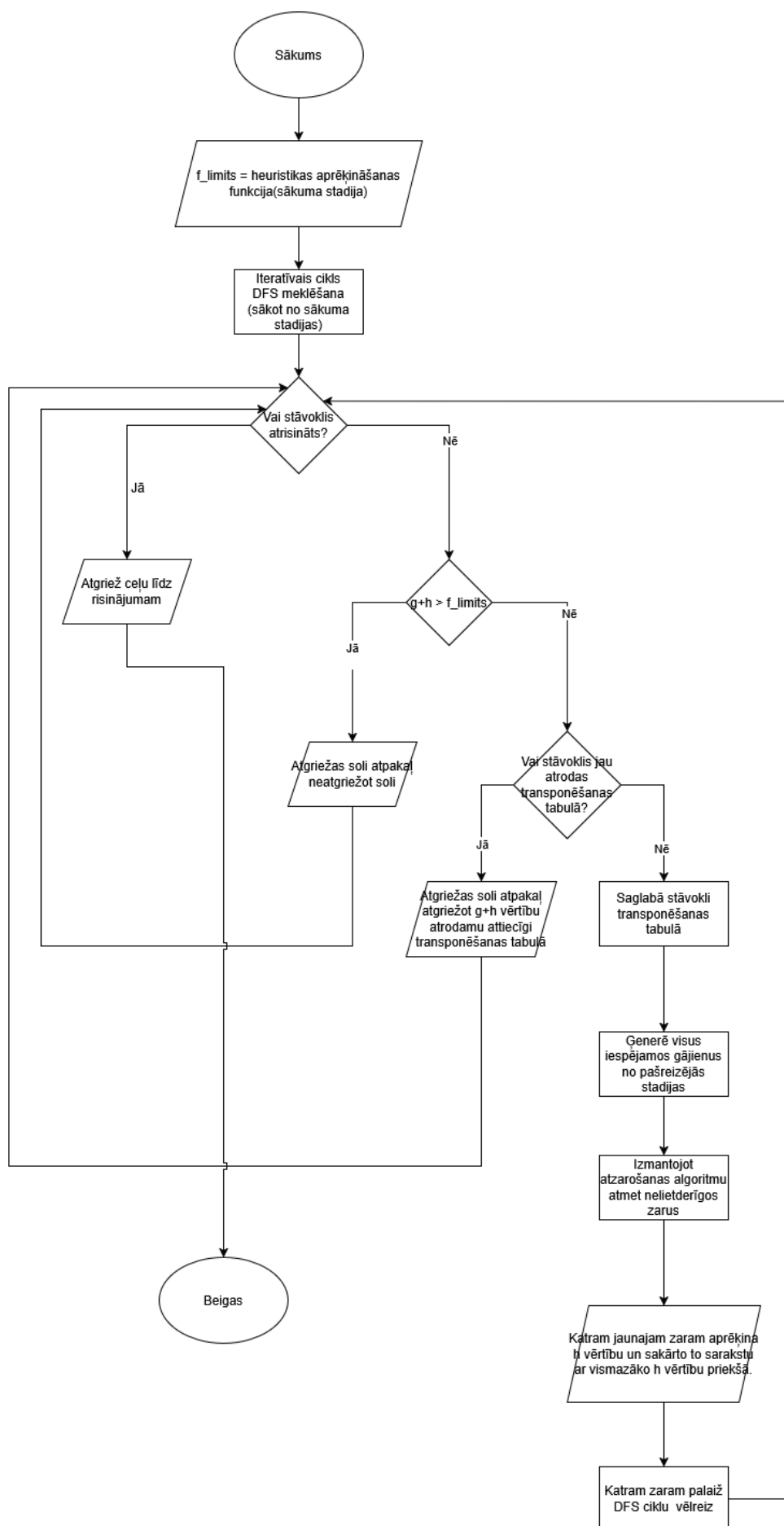
## IZMANTOTĀS INFORMĀCIJAS AVOTU SARAKSTS

1. Autentifikācijas nodrošinātājs - <https://appwrite.io/> (skatīts 02.03.2025)
2. <https://github.com/meltwater/served> - HTTP REST servera nodrošināšana C++ vidē (skatīts 02.03.2025)
3. <https://js.cubing.net/cubing/twisty/> - Rubika kuba vizualizācijas rīks (skatīts 03.05.2025)
4. <https://kociemba.org/cube.htm> - informācija par to kā tika izveidots Kociembas algoritms (skatīts 10.05.2025)
5. <https://nodejs.org/en> - starpniekservera serviss (skatīts 05.05.2025)
6. <https://www.drawio.com/> - diagrammu izveidošanas rīks (skatīts 12.05.2025)
7. <https://www.figma.com/> - saskarņu veidošanas rīks (skatīts 17.05.2025)
8. <https://www.mongodb.com/products/platform/atlas-database> - datubāzes viesošanās pakalpojumu sniedzējs. (skatīts 21.05.2025)
9. <https://www.speedcubedb.com/> - algoritmu datubāze (skatīts 22.05.2025)



## **PIELIKUMI**





## Pielikums Nr.3



# IZSTRĀDĀTĀS PROGRAMMAS PIRMTEKSTS

## cubieCube.cpp

Šis C++ kods definē Rubika kuba iekšējo stāvokļa reprezentāciju, kas seko līdzī stūru un malu permutācijām un orientācijām. Kods satur arī iepriekš aprēķinātas pārvietošanas tabulas (move tables) konkrētām kubu griešanas darbībām. Tas ir pamats IDA\* algoritmam.

```
#include <set>
#include <cstdint>
#include <iostream>
#include <array>
#include <array>
#include "pieces.h"
#include <string>
#include <algorithm>
#include <vector>

// Move tables for U face (90, 180, 270 degrees)
const std::array<Corner, 8> _cpU = {
    Corner::UBR, Corner::URF, Corner::UFL, Corner::ULB,
    Corner::DFR, Corner::DLF, Corner::DBL, Corner::DRB
};

const std::array<int, 8> _coU = {0, 0, 0, 0, 0, 0, 0, 0};

const std::array<Edge, 12> _epU = {
    Edge::UB, Edge::UR, Edge::UF,
    Edge::UL, Edge::DR, Edge::DF,
    Edge::DL, Edge::DB, Edge::FR,
    Edge::FL, Edge::BL, Edge::BR
};

const std::array<int, 12> _eoU = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};

const std::array<Corner, 8> _cpU2 = {
    Corner::ULB, Corner::UBR, Corner::URF, Corner::UFL,
    Corner::DFR, Corner::DLF, Corner::DBL, Corner::DRB
};

const std::array<int, 8> _coU2 = {0, 0, 0, 0, 0, 0, 0, 0};

const std::array<Edge, 12> _epU2 = {
    Edge::UL, Edge::UB, Edge::UR, Edge::UF,
    Edge::DR, Edge::DF, Edge::DL, Edge::DB,
    Edge::FR, Edge::FL, Edge::BL, Edge::BR
};

const std::array<int, 12> _eoU2 = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};

const std::array<Corner, 8> _cpUP = {
```

```

    Corner::UFL, Corner::ULB, Corner::UBR, Corner::URF,
    Corner::DFR, Corner::DLF, Corner::DBL, Corner::DRB
};

const std::array<int, 8> _coUP = {0, 0, 0, 0, 0, 0, 0, 0};

const std::array<Edge, 12> _epUP = {
    Edge::UF, Edge::UL, Edge::UB, Edge::UR,
    Edge::DR, Edge::DF, Edge::DL, Edge::DB,
    Edge::FR, Edge::FL, Edge::BL, Edge::BR
};

const std::array<int, 12> _eoUP = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};

// Move tables for R face
const std::array<Corner, 8> _cpR = {
    Corner::DFR, Corner::UFL, Corner::ULB, Corner::URF,
    Corner::DRB, Corner::DLF, Corner::DBL, Corner::UBR
};

const std::array<int, 8> _coR = {2, 0, 0, 1, 1, 0, 0, 2};

const std::array<Edge, 12> _epR = {
    Edge::FR, Edge::UF, Edge::UL, Edge::UB,
    Edge::BR, Edge::DF, Edge::DL, Edge::DB,
    Edge::DR, Edge::FL, Edge::BL, Edge::UR
};

const std::array<int, 12> _eoR = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};

// Rest of the rotations for R, F, D, L, B, U would be defined similarly

// List of all corners in solved order
const std::array<Corner, 8> _corners = {
    Corner::URF,
    Corner::UFL,
    Corner::ULB,
    Corner::UBR,
    Corner::DFR,
    Corner::DLF,
    Corner::DBL,
    Corner::DRB
};

// List of all edges in solved order
const std::array<Edge, 12> _edges = {
    Edge::UR,
    Edge::UF,
    Edge::UL,
    Edge::UB,
    Edge::DR,

```

```

    Edge::DF,
    Edge::DL,
    Edge::DB,
    Edge::FR,
    Edge::FL,
    Edge::BL,
    Edge::BR
};

// Set of edges that form the cross
const std::set<Edge> _edgesCross_set = {Edge::UF, Edge::UR, Edge::UL,
Edge::UB};

// Main cube representation class
class CubieCube {
public:
    // Constructor: optionally initialize with specific corners/edges
    CubieCube(const std::vector<Corner>& corners = {}, const
std::vector<Edge>& edges = {}) {
        // Initialize all corners and orientations to solved state
        this->cp = {Corner::URF, Corner::UFL, Corner::ULB, Corner::UBR,
Corner::DFR, Corner::DLF, Corner::DBL, Corner::DRB};
        this->co = {0, 0, 0, 0, 0, 0, 0, 0};
        if (!corners.empty()) {
            // If corners provided, set cpf/cof for F2L tracking
            for (int i = 0; i < 8; ++i) {
                this->cpf[i] = (std::find(corners.begin(), corners.end(),
_corners[i]) != corners.end()) ? _corners[i] : Corner(-1);
                this->cof[i] = (std::find(corners.begin(), corners.end(),
_corners[i]) != corners.end()) ? 0 : -1;
            }
            this->corners = corners;
        } else {
            // Default: only first 4 corners are valid
            this->cpf = {Corner::URF, Corner::UFL, Corner::ULB, Corner::UBR,
Corner(-1), Corner(-1), Corner(-1), Corner(-1)};
            this->cof = {0, 0, 0, 0, -1, -1, -1, -1};
            this->corners = {Corner::URF, Corner::UFL, Corner::ULB,
Corner::UBR};
        }

        // Initialize all edges and orientations to solved state
        this->ep = {Edge::UR, Edge::UF, Edge::UL, Edge::UB, Edge::DR,
Edge::DF, Edge::DL, Edge::DB, Edge::FR, Edge::FL, Edge::BL, Edge::BR};
        this->eo = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
        this->epc = {Edge::UR, Edge::UF, Edge::UL, Edge::UB, Edge(-1), Edge(-
1), Edge(-1), Edge(-1), Edge(-1), Edge(-1), Edge(-1), Edge(-1)};
        this->eoc = {0, 0, 0, 0, -1, -1, -1, -1, -1, -1, -1, -1};

        if (!edges.empty()) {
            // If edges provided, set epf/eof for F2L tracking

```

```

        for (int i = 0; i < 12; ++i) {
            this->epf[i] = (std::find(edges.begin(), edges.end(),
            _edges[i]) != edges.end()) ? _edges[i] : Edge(-1);
            this->eof[i] = (std::find(edges.begin(), edges.end(),
            _edges[i]) != edges.end()) ? 0 : -1;
        }
        this->edges = edges;
    } else {
        // Default: only last 4 edges are valid
        this->epf = {Edge(-1), Edge(-1), Edge(-1), Edge(-1), Edge(-1),
        Edge(-1), Edge(-1), Edge(-1), Edge::FR, Edge::FL, Edge::BL, Edge::BR};
        this->eof = {-1, -1, -1, -1, -1, -1, -1, -1, 0, 0, 0, 0};
        this->edges = {Edge::BL, Edge::BR, Edge::FR, Edge::FL};
    }
}

// Copy constructor
CubieCube(const CubieCube& other) {
    *this = other;
}

// Multiply corner permutation and orientation by another cube (apply
move)
void corner_multiply(const CubieCube& b) {
    std::array<Corner, 8> corner_perm;
    std::array<int, 8> corner_ori;

    for (int i = 0; i < 8; ++i) {
        corner_perm[i] = cp[static_cast<int>(b.cp[i])];
        corner_ori[i] = (co[static_cast<int>(b.cp[i])] + b.co[i]) % 3;
    }

    cp = corner_perm;
    co = corner_ori;

    // Update F2L corner tracking
    std::array<Corner, 8> corner_f2l_perm = {Corner(-1), Corner(-1),
    Corner(-1), Corner(-1), Corner(-1), Corner(-1), Corner(-1), Corner(-1)};
    std::array<int, 8> corner_f2l_ori = {-1, -1, -1, -1, -1, -1, -1, -1};

    int f2l_i = 0;

    for (int i = 0; i < 8; ++i) {
        if (f2l_i == corners.size()) {
            break;
        }

        Corner cp_i = corner_perm[i];
        int co_i = corner_ori[i];
    }

```



```

        if (std::find(corners.begin(), corners.end(), cp_i) !=
corners.end()) {
            corner_f2l_perm[i] = cp_i;
            corner_f2l_ori[i] = co_i;
            ++f2l_i;
        }
    }

    cof = corner_f2l_ori;
    cpf = corner_f2l_perm;
}

// Multiply edge permutation and orientation by another cube (apply move)
void edge_multiply(const CubieCube& b) {
    std::array<Edge, 12> edge_perm;
    std::array<int, 12> edge_ori;

    for (int i = 0; i < 12; ++i) {
        edge_perm[i] = ep[static_cast<int>(b.ep[i])];
        edge_ori[i] = (eo[static_cast<int>(b.ep[i])] + b.eo[i]) % 2;
    }

    eo = edge_ori;
    ep = edge_perm;

    // Update cross and F2L edge tracking
    std::array<Edge, 12> edge_cross_perm = {Edge(-1), Edge(-1), Edge(-1),
Edge(-1), Edge(-1), Edge(-1), Edge(-1), Edge(-1), Edge(-1), Edge(-1),
Edge(-1), Edge(-1)};
    std::array<int, 12> edge_cross_ori = {-1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1};
    std::array<Edge, 12> edge_f2l_perm = {Edge(-1), Edge(-1), Edge(-1),
Edge(-1), Edge(-1), Edge(-1), Edge(-1), Edge(-1), Edge(-1), Edge(-1),
Edge(-1), Edge(-1)};
    std::array<int, 12> edge_f2l_ori = {-1, -1, -1, -1, -1, -1, -1, -1,
1, -1, -1, -1};

    int cross_i = 0;
    int f2l_i = 0;
    int self_edges_len = edges.size();

    for (int i = 0; i < 12; ++i) {
        if (cross_i == 4 && f2l_i == self_edges_len) {
            break;
        }

        Edge ep_i = edge_perm[i];
        int eo_i = edge_ori[i];

        if (_edgesCross_set.find(ep_i) != _edgesCross_set.end()) {
            edge_cross_perm[i] = ep_i;

```

```

        edge_cross_ori[i] = eo_i;
        ++cross_i;
    }

    if (std::find(edges.begin(), edges.end(), ep_i) != edges.end()) {
        edge_f2l_perm[i] = ep_i;
        edge_f2l_ori[i] = eo_i;
        ++f2l_i;
    }
}

eoc = edge_cross_ori;
epc = edge_cross_perm;
eof = edge_f2l_ori;
epf = edge_f2l_perm;
}

// Apply a move (as a CubieCube) to this cube
void move(CubieCube b) {
    corner_multiply(b);
    edge_multiply(b);
}

// Setters for cube state
void set_cp(const std::array<Corner, 8>& cp) {
    this->cp = cp;
}

void set_co(const std::array<int, 8>& co) {
    this->co = co;
}

void set_ep(const std::array<Edge, 12>& ep) {
    this->ep = ep;
}

// Rest of the set methods

// Getters for cube state
std::array<Corner, 8> get_cp() const {
    return cp;
}

std::array<int, 8> get_co() const {
    return co;
}

std::array<Edge, 12> get_ep() const {
    return ep;
}

```

```

        // Rest of the get methods

private:
    // Corner permutation and orientation
    std::array<Corner, 8> cp;
    std::array<int, 8> co;
    // Edge permutation and orientation
    std::array<Edge, 12> ep;
    std::array<int, 12> eo;
    // Cross and F2L tracking
    std::array<Edge, 12> epc;
    std::array<int, 12> eoc;
    std::array<Edge, 12> epf;
    std::array<int, 12> eof;
    std::vector<Corner> corners;
    std::vector<Edge> edges;
    std::array<Corner, 8> cpf;
    std::array<int, 8> cof;
};

// Initialize all move CubieCubes (U, R, F, D, L, B and their powers)
std::array<CubieCube, 18> initialize_move_cube() {
    std::array<CubieCube, 18> MOVE_CUBE;

    // U move
    MOVE_CUBE[0].set_cp(_cpU);
    MOVE_CUBE[0].set_co(_coU);
    MOVE_CUBE[0].set_ep(_epU);
    MOVE_CUBE[0].set_eo(_eoU);

    // R move
    MOVE_CUBE[1].set_cp(_cpR);
    MOVE_CUBE[1].set_co(_coR);
    MOVE_CUBE[1].set_ep(_epR);
    MOVE_CUBE[1].set_eo(_eoR);

    // F move (assume _cpF, _coF, _epF, _eoF are defined elsewhere)
    MOVE_CUBE[2].set_cp(_cpF);
    MOVE_CUBE[2].set_co(_coF);
    MOVE_CUBE[2].set_ep(_epF);
    MOVE_CUBE[2].set_eo(_eoF);

    // Rest of the moves initialization....

    return MOVE_CUBE;
}

```

## IDA\_star\_cross.cpp

Šis C++ kods realizē Rubika kuba krusta (cross) atrisināšanu, izmantojot IDA\* algoritmu ar iepriekš aprēķinātām heuristikas tabulām. Tas izmanto CubieCube klasi, lai pārvaldītu kuba stāvokli. Kods analizē kuba pēc iedotā sajaukuma (scramble), aprēķina heuristiku konkrētajam krustam un iteratīvi padziļina meklēšanu līdz atrod optimālāko risinājumu.

```
#include <iostream>
#include <vector>
#include <unordered_map>
#include <string>
#include <algorithm>
#include <cmath>
#include <limits>
#include <tuple>

#include "cubiecube.h"
#include "tableloader.h"
#include "pieces.h"

// Tableloader instance for heuristic tables
Tableloader tableLoader = Tableloader();

// Precomputed move cubes for each possible move
const std::array<CubieCube, 18> MOVE_CUBE = initialize_move_cube();

// Mapping from move index to move notation
const std::unordered_map<int, std::string> ACTIONS = {
    {0, "U"}, {1, "R"}, {2, "F"}, {3, "D"}, {4, "L"}, {5, "B"},
    {6, "U2"}, {7, "R2"}, {8, "F2"}, {9, "D2"}, {10, "L2"}, {11, "B2"},
    {12, "U'"}, {13, "R'"}, {14, "F'"}, {15, "D'"}, {16, "L'"}, {17, "B'"}
};

// Redundant actions to avoid consecutive moves on the same face
const std::unordered_map<int, std::vector<int>> REDUNDANT_ACTIONS = {
    {0, {0, 6, 12}}, {1, {1, 7, 13}}, {2, {2, 8, 14}}, {3, {3, 9, 15}},
    {4, {4, 10, 16}}, {5, {5, 11, 17}}, {6, {0, 6, 12}}, {7, {1, 7, 13}},
    {8, {2, 8, 14}}, {9, {3, 9, 15}}, {10, {4, 10, 16}}, {11, {5, 11, 17}},
    {12, {0, 6, 12}}, {13, {1, 7, 13}}, {14, {2, 8, 14}}, {15, {3, 9, 15}},
    {16, {4, 10, 16}}, {17, {5, 11, 17}}
};

// Redundant actions to avoid certain move sequences (e.g., face turns on
// opposite faces)
const std::unordered_map<int, std::vector<int>> REDUNDANT_ACTIONS_2 = {
    {0, {3, 9, 15}}, {1, {4, 10, 16}}, {2, {5, 11, 17}}, {3, {0, 6, 12}},
    {4, {1, 7, 13}}, {5, {2, 8, 14}}, {6, {3, 9, 15}}, {7, {4, 10, 16}},
    {8, {5, 11, 17}}, {9, {0, 6, 12}}, {10, {1, 7, 13}}, {11, {2, 8, 14}},
    {12, {3, 9, 15}}, {13, {4, 10, 16}}, {14, {5, 11, 17}}, {15, {0, 6, 12}},
```

```

        {16, {1, 7, 13}}, {17, {2, 8, 14}}
};

// Check if the current cross state matches the goal
bool is_goal_state(std::string crossState, std::string goal_cross_state) {
    return crossState == goal_cross_state;
}

// Apply a move (given as notation) to a cube and return the result
CubieCube actionsWithNotations(const std::string& action, CubieCube cube) {
    if(action == "U") cube.move(0, MOVE_CUBE[0]);
    else if(action == "R") cube.move(1, MOVE_CUBE[1]);
    else if(action == "F") cube.move(2, MOVE_CUBE[2]);
    else if(action == "D") cube.move(3, MOVE_CUBE[3]);
    else if(action == "L") cube.move(4, MOVE_CUBE[4]);
    else if(action == "B") cube.move(5, MOVE_CUBE[5]);
    else if(action == "U'") cube.move(12, MOVE_CUBE[12]);
    else if(action == "R'") cube.move(13, MOVE_CUBE[13]);
    else if(action == "F'") cube.move(14, MOVE_CUBE[14]);
    else if(action == "D'") cube.move(15, MOVE_CUBE[15]);
    else if(action == "L'") cube.move(16, MOVE_CUBE[16]);
    else if(action == "B'") cube.move(17, MOVE_CUBE[17]);
    else if(action == "U2") cube.move(6, MOVE_CUBE[6]);
    else if(action == "R2") cube.move(7, MOVE_CUBE[7]);
    else if(action == "F2") cube.move(8, MOVE_CUBE[8]);
    else if(action == "D2") cube.move(9, MOVE_CUBE[9]);
    else if(action == "L2") cube.move(10, MOVE_CUBE[10]);
    else if(action == "B2") cube.move(11, MOVE_CUBE[11]);
    return cube;
}

// Apply a sequence of moves (algorithm) to a cube
CubieCube do_algorithm(const std::string& algorithm, CubieCube cube) {
    std::istringstream iss(algorithm);
    std::string move;
    while (iss >> move) {
        cube = actionsWithNotations(move, cube);
    }
    return cube;
}

// IDA* search class for solving the cross
class IDA_star_cross {
public:
    // Constructor: sets max depth, goal state, and loads heuristics
    IDA_star_cross(int max_depth = 100){
        this->max_depth = max_depth;
        CubieCube cubeCheck = CubieCube();
        this->goal_cross_state = get_cross_state(cubeCheck);
        this->crossHeur = tableLoader.getHeuristics("cross");
    }

```

```

    // Run the IDA* search and return the solution as a sequence of move
indices
    std::vector<int> run(CubieCube cube) {
        int threshold = heuristic_value(cube);
        while (threshold <= max_depth) {
            std::cout << "Threshold: " << threshold << std::endl;
            moves.clear();
            transposition_table.clear();
            int distance = search(cube, 0, threshold);
            if (distance == 0) {
                std::cout << "Solution found" << std::endl;
                return moves;
            }
            if (distance == std::numeric_limits<int>::max()) {
                return {};
            }
            threshold = distance;
        }
        return {};
    }

private:
    int max_depth; // Maximum allowed search depth
    std::vector<int> moves; // Current path of moves
    std::unordered_map<std::string, std::pair<int, int>> transposition_table;
// Memoization for visited states
    std::string goal_cross_state; // Goal cross state string
    std::unordered_map<std::string, int> crossHeur; // Heuristic table for
cross states

    // Recursive search function for IDA*
    int search(CubieCube cube, int g_score, int threshold) {
        std::string cube_state = get_cube_state(cube);

        // Check if this state has already been visited with a lower g_score
        auto it = transposition_table.find(cube_state);
        if (it != transposition_table.end()) {
            auto [stored_g_score, stored_result] = it->second;
            if (stored_g_score <= g_score) {
                return stored_result;
            }
        }

        int f_score = g_score + heuristic_value(cube);
        if (f_score > threshold) {
            transposition_table[cube_state] = {g_score, f_score};
            return f_score;
        }

        // Check if goal state is reached

```

```

    if (is_goal_state(get_cross_state(cube), goal_cross_state)) {
        transposition_table[cube_state] = {g_score, 0};
        return 0;
    }

    int min_cost = std::numeric_limits<int>::max();
    // Try all possible moves
    for (int action = 0; action < 18; ++action) {
        CubieCube cube_copy = cube;
        cube_copy.move(action, MOVE_CUBE[action]);

        // Prune redundant moves (same face as previous move)
        if (!moves.empty() &&
std::find(REDUNDANT_ACTIONS.at(moves.back()).begin(),
                                                REDUNDANT_ACTIONS.at(moves.back())
.end(),
                                                action) !=
REDUNDANT_ACTIONS.at(moves.back()).end()) {
            continue;
        }

        // Prune more complex redundant move sequences
        if (moves.size() > 1 &&
            std::find(REDUNDANT_ACTIONS_2.at(moves.back()).begin(),
                    REDUNDANT_ACTIONS_2.at(moves.back()).end(),
                    action) !=
REDUNDANT_ACTIONS_2.at(moves.back()).end() &&
            std::find(REDUNDANT_ACTIONS.at(moves[moves.size() -
2]).begin(),
                    REDUNDANT_ACTIONS.at(moves[moves.size() - 2]).end(),
                    action) != REDUNDANT_ACTIONS.at(moves[moves.size() -
2]).end()) {
            continue;
        }

        moves.push_back(action);
        int distance = search(cube_copy, g_score + 1, threshold);
        if (distance == 0) {
            return 0;
        }
        if (distance < min_cost) {
            min_cost = distance;
        }
        moves.pop_back();
    }

    transposition_table[cube_state] = {g_score, min_cost};
    return min_cost;
}

```

```

// Get a string representation of the cube's cross state (used for
hashing)
std::string get_cube_state(const CubieCube& cube) {
    std::string state = get_cross_state(cube);
    return state;
}

// Extract the cross state from the cube as a string
std::string get_cross_state(const CubieCube& cube) {
    // (-1, 1, 2, 3, -1, -1, -1, -1, -1, -1, -1, 0, -1, 0, 0, 0, -1, -1, -
1, -1, -1, -1, -1, 0)
    std::string crossState = "(";
    std::array<Edge, 12> epc = cube.get_epc();
    std::array<int, 12> eoc = cube.get_eoc();

    // Encode edge positions for cross edges
    for(int i = 0; i < 12; i++){
        if(epc[i] == Edge::UF){
            crossState += "1, ";
        } else if(epc[i] == Edge::UL){
            crossState += "2, ";
        } else if(epc[i] == Edge::UB){
            crossState += "3, ";
        } else if(epc[i] == Edge::UR){
            crossState += "0, ";
        } else {
            crossState += "-1, ";
        }
    }

    // Encode edge orientations
    for(int i = 0; i < 12; i++){
        crossState += std::to_string(eoc[i]) + ", ";
    }

    crossState.pop_back();
    crossState.pop_back();
    crossState += ")";

    return crossState;
}

// Get the heuristic value for the current cross state
int heuristic_value(const CubieCube& cube) {
    std::string stateCross = get_cross_state(cube);

    auto it = crossHeur.find(stateCross);
    if (it != crossHeur.end()) {
        return it->second - 1;
    } else {
        return 5;
    }
}

```



```

    }
}
};

// Solve the cross for a given scramble and print the solution
std::vector<int> getCrossSolution(std::string scramble){
    CubieCube cube = CubieCube();
    cube = do_algorithm(scramble, cube);
    IDA_star_cross ida_star_cross = IDA_star_cross();
    std::vector<int> solution = ida_star_cross.run(cube);

    for (int action : solution) {
        std::cout << ACTIONS.at(action) << " ";
    }
    std::cout << std::endl;

    return solution;
};

```

## IDA\_star\_F2L.cpp

Šis C++ kods realizē Rubika kuba F2L soļa atrisināšanu, izmantojot IDA\* algoritmu ar iepriekš aprēķinātām heuristikas tabulām. Tas izmanto CubieCube klasi, lai pārvaldītu kuba stāvokli. Kods analizē kubu pēc iedotā sajaukuma (scramble) un iteratīvi padziļina meklēšanu līdz atrod optimālāko risinājumu.

```

#include <iostream>
#include <vector>
#include <unordered_map>
#include <string>
#include <algorithm>
#include <cmath>
#include <limits>
#include <tuple>
#include <map>

#include "cubiecube.h"
#include "tableloader.h"
#include "pieces.h"

// Initialize table loader and move cube array
Tableloader tableLoader = Tableloader();
const std::array<CubieCube, 18> MOVE_CUBE = initialize_move_cube();

// Mapping of move indices to move notations
const std::unordered_map<int, std::string> ACTIONS = {
    {0, "U"}, {1, "R"}, {2, "F"}, {3, "D"}, {4, "L"}, {5, "B"},
    {6, "U2"}, {7, "R2"}, {8, "F2"}, {9, "D2"}, {10, "L2"}, {11, "B2"},
    {12, "U'"}, {13, "R'"}, {14, "F'"}, {15, "D'"}, {16, "L'"}, {17, "B'"}
};

```

```

};

// Redundant moves for pruning (same face moves)
const std::unordered_map<int, std::vector<int>>> REDUNDANT_ACTIONS = {
    {0, {0, 6, 12}}, {1, {1, 7, 13}}, {2, {2, 8, 14}}, {3, {3, 9, 15}},
    {4, {4, 10, 16}}, {5, {5, 11, 17}}, {6, {0, 6, 12}}, {7, {1, 7, 13}},
    {8, {2, 8, 14}}, {9, {3, 9, 15}}, {10, {4, 10, 16}}, {11, {5, 11, 17}},
    {12, {0, 6, 12}}, {13, {1, 7, 13}}, {14, {2, 8, 14}}, {15, {3, 9, 15}},
    {16, {4, 10, 16}}, {17, {5, 11, 17}}
};

// Redundant moves for deeper pruning (opposite faces)
const std::unordered_map<int, std::vector<int>>> REDUNDANT_ACTIONS_2 = {
    {0, {3, 9, 15}}, {1, {4, 10, 16}}, {2, {5, 11, 17}}, {3, {0, 6, 12}},
    {4, {1, 7, 13}}, {5, {2, 8, 14}}, {6, {3, 9, 15}}, {7, {4, 10, 16}},
    {8, {5, 11, 17}}, {9, {0, 6, 12}}, {10, {1, 7, 13}}, {11, {2, 8, 14}},
    {12, {3, 9, 15}}, {13, {4, 10, 16}}, {14, {5, 11, 17}}, {15, {0, 6, 12}},
    {16, {1, 7, 13}}, {17, {2, 8, 14}}
};

// Check if the cube is in the goal state
bool is_goal_state(std::string crossState, std::string goal_cross_state,
std::string goal_edge_stae, std::string goal_corner_state, std::string
edge_state, std::string corner_state) {
    return crossState == goal_cross_state && edge_state == goal_edge_stae &&
corner_state == goal_corner_state;
}

// Apply a move to a cube given its notation
CubieCube actionsWithNotations(const std::string& action, CubieCube cube) {
    if(action == "U") cube.move(0, MOVE_CUBE[0]);
    else if(action == "R") cube.move(1, MOVE_CUBE[1]);
    else if(action == "F") cube.move(2, MOVE_CUBE[2]);
    else if(action == "D") cube.move(3, MOVE_CUBE[3]);
    else if(action == "L") cube.move(4, MOVE_CUBE[4]);
    else if(action == "B") cube.move(5, MOVE_CUBE[5]);
    else if(action == "U'") cube.move(12, MOVE_CUBE[12]);
    else if(action == "R'") cube.move(13, MOVE_CUBE[13]);
    else if(action == "F'") cube.move(14, MOVE_CUBE[14]);
    else if(action == "D'") cube.move(15, MOVE_CUBE[15]);
    else if(action == "L'") cube.move(16, MOVE_CUBE[16]);
    else if(action == "B'") cube.move(17, MOVE_CUBE[17]);
    else if(action == "U2") cube.move(6, MOVE_CUBE[6]);
    else if(action == "R2") cube.move(7, MOVE_CUBE[7]);
    else if(action == "F2") cube.move(8, MOVE_CUBE[8]);
    else if(action == "D2") cube.move(9, MOVE_CUBE[9]);
    else if(action == "L2") cube.move(10, MOVE_CUBE[10]);
    else if(action == "B2") cube.move(11, MOVE_CUBE[11]);
    return cube;
}

```

```

// Apply a sequence of moves (algorithm) to a cube
CubieCube do_algorithm(const std::string& algorithm, CubieCube cube) {
    std::istringstream iss(algorithm);
    std::string move;
    while (iss >> move) {
        cube = actionsWithNotations(move, cube);
    }
    return cube;
}

// IDA* search class for F2L
class IDA_star_F2L {
public:
    // Constructor initializes goal states and heuristic tables
    IDA_star_F2L(std::vector<Corner> corners, std::vector<Edge> edges,
std::string cornerStr, std::string edgeStr, int max_depth = 100){
        this->max_depth = max_depth;
        CubieCube cubeCheck = CubieCube(corners, edges);
        this->goal_cross_state = get_cross_state(cubeCheck);
        this->goal_corner_state = get_corner_state(cubeCheck);
        this->goal_edge_stae = get_edge_state(cubeCheck);
        this->crossHeur = tableLoader.getHeuristics("cross");
        this->cornerHeur = tableLoader.getHeuristics(cornerStr);
        this->edgeHeur = tableLoader.getHeuristics(edgeStr);
        this->corners = corners;
        this->edges = edges;
    }

    // Run the IDA* search and return the solution moves
    std::vector<int> run(CubieCube cube) {
        int threshold = heuristic_value(cube);
        while (threshold <= max_depth) {
            std::cout << "Threshold: " << threshold << '\n';
            moves.clear();
            transposition_table.clear();
            int distance = search(cube, 0, threshold);
            if (distance == 0) {
                std::cout << "Solution found" << '\n';
                return moves;
            }
            if (distance == std::numeric_limits<int>::max()) {
                return {};
            }
            threshold = distance;
        }
        return {};
    }

private:
    int max_depth;
    std::vector<int> moves;

```

```

std::unordered_map<std::string, std::pair<int, int>> transposition_table;
std::string goal_cross_state;
std::string goal_corner_state;
std::string goal_edge_state;
std::unordered_map<std::string, int> crossHeur;
std::unordered_map<std::string, int> cornerHeur;
std::unordered_map<std::string, int> edgeHeur;
std::vector<Corner> corners;
std::vector<Edge> edges;

// Recursive search function for IDA*
int search(CubieCube cube, int g_score, int threshold) {
    std::string cube_state = get_cube_state(cube);

    // Transposition table lookup
    auto it = transposition_table.find(cube_state);
    if (it != transposition_table.end()) {
        auto [stored_g_score, stored_result] = it->second;
        if (stored_g_score <= g_score) {
            return stored_result;
        }
    }

    int f_score = g_score + heuristic_value(cube);
    if (f_score > threshold) {
        transposition_table[cube_state] = {g_score, f_score};
        return f_score;
    }

    // Check for goal state
    if (is_goal_state(get_cross_state(cube), goal_cross_state,
goal_edge_state, goal_corner_state, get_edge_state(cube),
get_corner_state(cube))) {
        transposition_table[cube_state] = {g_score, 0};
        return 0;
    }

    int min_cost = std::numeric_limits<int>::max();
    for (int action = 0; action < 18; ++action) {
        CubieCube cube_copy = cube;
        cube_copy.move(action, MOVE_CUBE[action]);

        // Prune redundant moves (same face)
        if (!moves.empty() &&
std::find(REDUNDANT_ACTIONS.at(moves.back()).begin(),
REDUNDANT_ACTIONS.at(moves.back())
.end(),
action) !=
REDUNDANT_ACTIONS.at(moves.back()).end()) {
            continue;
        }
    }

```

```

        // Prune deeper redundant moves (opposite faces)
        if (moves.size() > 1 &&
            std::find(REDUNDANT_ACTIONS_2.at(moves.back()).begin(),
                    REDUNDANT_ACTIONS_2.at(moves.back()).end(),
                    action) !=
REDUNDANT_ACTIONS_2.at(moves.back()).end() &&
            std::find(REDUNDANT_ACTIONS.at(moves[moves.size() -
2]).begin(),
                    REDUNDANT_ACTIONS.at(moves[moves.size() - 2]).end(),
                    action) != REDUNDANT_ACTIONS.at(moves[moves.size() -
2]).end()) {
            continue;
        }

        moves.push_back(action);
        int distance = search(cube_copy, g_score + 1, threshold);
        if (distance == 0) {
            return 0;
        }
        if (distance < min_cost) {
            min_cost = distance;
        }
        moves.pop_back();
    }

    transposition_table[cube_state] = {g_score, min_cost};
    return min_cost;
}

// Get a string representing the cube state (for hashing)
std::string get_cube_state(const CubieCube& cube) {
    std::string state = get_cross_state(cube) + " " + get_edge_state(cube)
+ " " + get_corner_state(cube);
    return state;
}

// Get cross state as a string
std::string get_cross_state(const CubieCube& cube) {
    // (-1, 1, 2, 3, -1, -1, -1, -1, -1, -1, -1, 0, -1, 0, 0, 0, -1, -1, -
1, -1, -1, -1, -1, 0)
    std::string crossState = "(";
    std::array<Edge, 12> epc = cube.get_epc();
    std::array<int, 12> eoc = cube.get_eoc();

    for(int i = 0; i < 12; i++){
        if(epc[i] == Edge::UF){
            crossState += "1, ";
        } else if(epc[i] == Edge::UL){
            crossState += "2, ";
        } else if(epc[i] == Edge::UB){

```

```

        crossState += "3, ";
    } else if(epc[i] == Edge::UR){
        crossState += "0, ";
    } else {
        crossState += "-1, ";
    }
}

for(int i = 0; i < 12; i++){
    crossState += std::to_string(eoc[i]) + ", ";
}

crossState.pop_back();
crossState.pop_back();
crossState += ")";

return crossState;
}

// Get edge state as a string
std::string get_edge_state(const CubieCube& cube) {
    std::string edgeState = "(";
    std::array<Edge, 12> epf = cube.get_epf();
    std::array<int, 12> eof = cube.get_eof();

    for(int i = 0; i < 12; i++){
        if(epf[i] == Edge::FR){
            edgeState += "8, ";
        } else if(epf[i] == Edge::BR){
            edgeState += "11, ";
        } else if(epf[i] == Edge::BL){
            edgeState += "10, ";
        } else if(epf[i] == Edge::FL){
            edgeState += "9, ";
        } else {
            edgeState += "-1, ";
        }
    }

    for(int i = 0; i < 12; i++){
        edgeState += std::to_string(eof[i]) + ", ";
    }

    edgeState.pop_back();
    edgeState.pop_back();
    edgeState += ")";

    return edgeState;
}

// Get corner state as a string

```

```

std::string get_corner_state(const CubieCube& cube) {
    std::string cornerState = "(";
    std::array<Corner, 8> cpf = cube.get_cpf();
    std::array<int, 8> cof = cube.get_cof();

    for(int i = 0; i < 8; i++){
        if(cpf[i] == Corner::URF){
            cornerState += "0, ";
        } else if(cpf[i] == Corner::UFL){
            cornerState += "1, ";
        } else if(cpf[i] == Corner::UBR){
            cornerState += "3, ";
        } else if(cpf[i] == Corner::ULB){
            cornerState += "2, ";
        } else {
            cornerState += "-1, ";
        }
    }

    for(int i = 0; i < 8; i++){
        cornerState += std::to_string(cof[i]) + ", ";
    }

    cornerState.pop_back();
    cornerState.pop_back();
    cornerState += ")";

    return cornerState;
}

// Heuristic value for the current cube state
int heuristic_value(const CubieCube& cube) {
    std::string stateCross = get_cross_state(cube);
    std::string stateEdge = get_edge_state(cube);
    std::string stateCorner = get_corner_state(cube);

    auto itCross = crossHeur.find(stateCross);
    auto itEdge = edgeHeur.find(stateEdge);
    auto itCorner = cornerHeur.find(stateCorner);

    int h_cross = 5, h_edge = 5, h_corner = 5;

    if (itCross != crossHeur.end()) {
        h_cross = crossHeur[stateCross] - 1;
    }
    if(itEdge != edgeHeur.end()){
        h_edge = edgeHeur[stateEdge];
    }
    if(itCorner != cornerHeur.end()){
        h_corner = cornerHeur[stateCorner];
    }
}

```

```

        return h_cross + h_edge + h_corner;
    }
};

// Try all F2L pair orders and return solutions for each
std::vector<std::vector<std::vector<int>>> getSolutions(std::string scramble,
std::string crossSolution){
    CubieCube cube = CubieCube();
    cube = do_algorithm(scramble, cube);

    std::vector<Corner> solved_f2l_corners = {};
    std::vector<Edge> solved_f2l_edges = {};

    std::array<Corner, 4> f2l_corners = {Corner::URF, Corner::UFL,
Corner::ULB, Corner::UBR};
    std::array<Edge, 4> f2l_edges = {Edge::FR, Edge::FL, Edge::BL, Edge::BR};

    std::vector<std::array<Corner, 4>> f2l_corners_combinations;
    std::vector<std::array<Edge, 4>> f2l_edges_combinations;

    // Generate all permutations of F2L pairs
    std::array<Corner, 4> corners = {Corner::URF, Corner::UFL, Corner::ULB,
Corner::UBR};
    do {
        f2l_corners_combinations.push_back(corners);
    } while (std::next_permutation(corners.begin(), corners.end()));

    std::array<Edge, 4> edges = {Edge::FR, Edge::FL, Edge::BL, Edge::BR};
    do {
        f2l_edges_combinations.push_back(edges);
    } while (std::next_permutation(edges.begin(), edges.end()));

    std::vector<std::vector<std::vector<int>>> all_sol = {};
    std::map<std::array<std::string, 6>, std::vector<int>> sol_heur = {};

    int startTime_main = clock();
    // For each permutation, solve F2L pairs in order
    for(int j = 0; j < f2l_corners_combinations.size(); j++){
        std::vector<std::vector<int>> f2l_sol = {};
        std::array<Corner, 4> f2l_corners = f2l_corners_combinations[j];
        std::array<Edge, 4> f2l_edges = f2l_edges_combinations[j];

        for(int i = 0; i < f2l_corners.size(); i++){
            std::cout << "F2L " << i + 1 << '\n';

            std::vector<Corner> corners = {};
            std::vector<Edge> edges = {};

            // Build up the list of solved pairs
            for(int k = 0; k <= i; k++){
                corners.push_back(f2l_corners[k]);
            }
        }
    }
}

```



```

        edges.push_back(f2l_edges[k]);
    }

    cube = CubieCube(corners, edges);
    cube = do_algorithm(scramble, cube);
    cube = do_algorithm(crossSolution, cube);

    // Apply previous F2L solutions
    for(int k = 0; k < f2l_sol.size() ; k++){
        for(int l = 0; l < f2l_sol[k].size(); l++){
            cube.move(f2l_sol[k][l], MOVE_CUBE[f2l_sol[k][l]]);
        }
    }

    std::vector<int> cornerStr, edgeStr;

    for (Corner corner : corners) {
        cornerStr.push_back(static_cast<int>(corner));
    }
    for (Edge edge : edges) {
        edgeStr.push_back(static_cast<int>(edge));
    }

    std::sort(cornerStr.begin(), cornerStr.end());
    std::sort(edgeStr.begin(), edgeStr.end());

    std::string cornerStrJoined;
    std::string edgeStrJoined;
    for (int val : cornerStr) {
        cornerStrJoined += std::to_string(val);
    }
    for (int val : edgeStr) {
        edgeStrJoined += std::to_string(val);
    }

    std::cout << cornerStrJoined << '\n';
    std::cout << edgeStrJoined << '\n';

    // Build key for memoization
    std::string cofToString = "";
    std::string eofToString = "";
    std::string cpfToString = "";
    std::string epfToString = "";
    for(Corner corner: cube.get_cpf()){
        cpfToString += std::to_string(static_cast<int>(corner));
    }
    for(int val: cube.get_cof()){
        cofToString += std::to_string(val);
    }
    for(Edge edge: cube.get_epf()){
        epfToString += std::to_string(static_cast<int>(edge));
    }

```

```

    }
    for(int val: cube.get_eof()){
        eofToString += std::to_string(val);
    }

    std::array<std::string, 6> key = {cornerStrJoined, edgeStrJoined,
    cofToString, eofToString, cpfToString, epfToString};

    // Use memoized solution if available
    if(sol_heur.find(key) != sol_heur.end()){
        f2l_sol.push_back(sol_heur[key]);
        continue;
    }

    // Run IDA* for this F2L pair
    IDA_star_F2L ida_star_f2l = IDA_star_F2L(corners, edges,
    cornerStrJoined, edgeStrJoined);
    int startTime = clock();
    std::vector<int> moves = ida_star_f2l.run(cube);

    f2l_sol.push_back(moves);
    int endTime = clock();

    std::cout << "Time: " << (endTime - startTime) / (double)
    CLOCKS_PER_SEC << '\n';

    sol_heur[key] = moves;
}
std::cout << "Scramble: " << scramble << '\n';
std::cout << "Cross solution: " << crossSolution << '\n';

// Print solution for this permutation
for(std::vector<int> alg : f2l_sol){
    for(int move : alg){
        std::cout << ACTIONS.at(move) << " ";
    }
    std::cout << '\n';
}

all_sol.push_back(f2l_sol);
}
int endTime_main = clock();
std::cout << "COMPLETE TIME: " << (endTime_main - startTime_main) /
(double) CLOCKS_PER_SEC << '\n';

return all_sol;
}

// Solve F2L pairs in a specific order (given by 'order')
std::vector<std::vector<std::vector<int>>> getSolutionsOneByOne(std::string
scramble, std::string crossSolution, std::vector<int> order){

```

```

CubieCube cube = CubieCube();
cube = do_algorithm(scramble, cube);

std::vector<Corner> solved_f2l_corners = {};
std::vector<Edge> solved_f2l_edges = {};

std::array<Corner, 4> f2l_corners = {Corner::URF, Corner::UFL,
Corner::ULB, Corner::UBR};
std::array<Edge, 4> f2l_edges = {Edge::FR, Edge::FL, Edge::BL, Edge::BR};

std::vector<std::array<Corner, 4>> f2l_corners_combinations;
std::vector<std::array<Edge, 4>> f2l_edges_combinations;

std::array<Corner, 4> corners = {Corner::URF, Corner::UFL, Corner::ULB,
Corner::UBR};
std::array<Edge, 4> edges = {Edge::FR, Edge::FL, Edge::BL, Edge::BR};

std::array<Corner, 4> ordered_corners;
std::array<Edge, 4> ordered_edges;

// Apply the given order to corners and edges
for (int i = 0; i < 4; ++i) {
    ordered_corners[i] = corners[order[i]];
    ordered_edges[i] = edges[order[i]];
}

f2l_corners_combinations.push_back(ordered_corners);
f2l_edges_combinations.push_back(ordered_edges);

std::vector<std::vector<std::vector<int>>> all_sol = {};
std::map<std::array<std::string, 6>, std::vector<int>> sol_heur = {};

int startTime_main = clock();
// Only one permutation (the given order)
for(int j = 0; j < f2l_corners_combinations.size(); j++){
    std::vector<std::vector<int>> f2l_sol = {};
    std::array<Corner, 4> f2l_corners = f2l_corners_combinations[j];
    std::array<Edge, 4> f2l_edges = f2l_edges_combinations[j];

    for(int i = 0; i < f2l_corners.size(); i++){
        std::cout << "F2L " << i + 1 << '\n';

        std::vector<Corner> corners = {};
        std::vector<Edge> edges = {};

        // Build up the list of solved pairs
        for(int k = 0; k <= i; k++){
            corners.push_back(f2l_corners[k]);
            edges.push_back(f2l_edges[k]);
        }
    }
}

```

```

cube = CubieCube(corners, edges);
cube = do_algorithm(scramble, cube);
cube = do_algorithm(crossSolution, cube);

// Apply previous F2L solutions
for(int k = 0; k < f2l_sol.size() ; k++){
    for(int l = 0; l < f2l_sol[k].size(); l++){
        cube.move(f2l_sol[k][l], MOVE_CUBE[f2l_sol[k][l]]);
    }
}

std::vector<int> cornerStr, edgeStr;

for (Corner corner : corners) {
    cornerStr.push_back(static_cast<int>(corner));
}
for (Edge edge : edges) {
    edgeStr.push_back(static_cast<int>(edge));
}

std::sort(cornerStr.begin(), cornerStr.end());
std::sort(edgeStr.begin(), edgeStr.end());

std::string cornerStrJoined;
std::string edgeStrJoined;
for (int val : cornerStr) {
    cornerStrJoined += std::to_string(val);
}
for (int val : edgeStr) {
    edgeStrJoined += std::to_string(val);
}

std::cout << cornerStrJoined << '\n';
std::cout << edgeStrJoined << '\n';

// Build key for memoization
std::string cofToString = "";
std::string eofToString = "";
std::string cpfToString = "";
std::string epfToString = "";
for(Corner corner: cube.get_cpf()){
    cpfToString += std::to_string(static_cast<int>(corner));
}
for(int val: cube.get_cof()){
    cofToString += std::to_string(val);
}
for(Edge edge: cube.get_epf()){
    epfToString += std::to_string(static_cast<int>(edge));
}
for(int val: cube.get_eof()){
    eofToString += std::to_string(val);
}

```

```

    }

    std::array<std::string, 6> key = {cornerStrJoined, edgeStrJoined,
    cofToString, eofToString, cpfToString, epfToString};

    // Use memoized solution if available
    if(sol_heur.find(key) != sol_heur.end()){
        f2l_sol.push_back(sol_heur[key]);
        continue;
    }

    // Run IDA* for this F2L pair
    IDA_star_F2L ida_star_f2l = IDA_star_F2L(corners, edges,
    cornerStrJoined, edgeStrJoined);
    int startTime = clock();
    std::vector<int> moves = ida_star_f2l.run(cube);

    f2l_sol.push_back(moves);
    int endTime = clock();

    std::cout << "Time: " << (endTime - startTime) / (double)
    CLOCKS_PER_SEC << '\n';

    sol_heur[key] = moves;
}
std::cout << "Scramble: " << scramble << '\n';
std::cout << "Cross solution: " << crossSolution << '\n';

// Print solution for this order
for(std::vector<int> alg : f2l_sol){
    for(int move : alg){
        std::cout << ACTIONS.at(move) << " ";
    }
    std::cout << '\n';
}

all_sol.push_back(f2l_sol);
}
int endTime_main = clock();
std::cout << "COMPLETE TIME: " << (endTime_main - startTime_main) /
(double) CLOCKS_PER_SEC << '\n';

return all_sol;
}

// Main function: runs both getSolutions and getSolutionsOneByOne
int main(){
    std::string scramble = "D' F2 U2 L B D' R' F' L U' F' L' U' B2 D F2 R D L
    F U' L' F";
    std::string crossSolution = "F B' R F D B2";

```

```

    std::vector<std::vector<std::vector<int>>> all_sol =
    getSolutions(scramble, crossSolution);

    std::cout << "Solutions from getSolutions: " << '\n';
    for(std::vector<std::vector<int>> f2l_sol : all_sol){
        for(std::vector<int> alg : f2l_sol){
            for(int move : alg){
                std::cout << ACTIONS.at(move) << " ";
            }
            std::cout << '\n';
        }
        std::cout << '\n';
    }

    std::vector<int> order = {0, 1, 2, 3};
    std::vector<std::vector<std::vector<int>>> all_sol_one_by_one =
    getSolutionsOneByOne(scramble, crossSolution, order);

    std::cout << "Solutions from getSolutionsOneByOne: " << '\n';
    for(std::vector<std::vector<int>> f2l_sol : all_sol_one_by_one){
        for(std::vector<int> alg : f2l_sol){
            for(int move : alg){
                std::cout << ACTIONS.at(move) << " ";
            }
            std::cout << '\n';
        }
        std::cout << '\n';
    }

    return 0;
};

```

### cpprestserver.cpp

Šis C++ kods izveido REST API serveri ar diviem galvenajiem maršrutiem. Abi maršruti pieņem Rubika kuba "scramble" (sajaukšanas secību), aprēķina krusta atrisinājumu un F2L (First Two Layers) atrisinājumus, izmantojot IDA\* algoritmu. Rezultāts tiek formatēts JSON formātā un atgriezts klientam. Otrajā maršrutā lietotājs var arī norādīt F2L pāru kārtošanas secību.

```

#include "idaStarF2l.h"
#include "idaStarCross.h"

#include <iostream>
#include <served/served.hpp>
#include <future>

// Function to process the solution for the /api/v1/greeting endpoint
// Takes a scramble string, computes the cross solution and all F2L solutions,

```

```

// and formats the result as a JSON string.
std::string process_solution(const std::string &scramble) {
    // Get cross solution as a vector of moves
    std::vector<int> crossSol = getCrossSolution(scramble);

    // Convert cross solution moves to string
    std::string crossSolution;
    for (int move : crossSol) {
        crossSolution += ACTIONS.at(move) + " ";
    }

    // Get all F2L solutions based on the scramble and cross solution
    std::vector<std::vector<std::vector<int>>> allSol = getSolutions(scramble,
crossSolution);
    std::ostream response;

    // Begin JSON response
    response << "{\n";
    response << "  \"cross_solution\":[\n";
    response << "    \"";
    for (int move : crossSol) {
        response << ACTIONS.at(move) << " ";
    }
    response << "\"\n";
    response << "  ],\n";

    // Add all F2L solutions to the JSON response
    int counter = 0;
    for (const auto &f2l_sol : allSol) {
        response << "  \"solution\" << counter++ << "\":\n";
        for (size_t i = 0; i < f2l_sol.size(); ++i) {
            response << "    \"";
            for (int move : f2l_sol[i]) {
                response << ACTIONS.at(move) << " ";
            }
            response << "\"";
            if (i != f2l_sol.size() - 1) {
                response << ",";
            }
            response << "\n";
        }
        response << "  ]";
        if (counter != allSol.size()) {
            response << ",";
        }
        response << "\n";
    }

    response << "}\n";
    return response.str();
}

```

```

// Function to process the solution for the /api/v1/greeting_one_by_one
// endpoint
// Takes a scramble string and an order vector, computes the cross solution
// and all F2L solutions in the given order,
// and formats the result as a JSON string.
std::string process_solution_one_by_one(const std::string &scramble, const
std::vector<int> &order) {
    // Get cross solution as a vector of moves
    std::vector<int> crossSol = getCrossSolution(scramble);

    // Convert cross solution moves to string
    std::string crossSolution;
    for (int move : crossSol) {
        crossSolution += ACTIONS.at(move) + " ";
    }

    // Get all F2L solutions in the specified order
    std::vector<std::vector<std::vector<int>>> allSol =
getSolutionsOneByOne(scramble, crossSolution, order);
    std::ostringstream response;

    // Begin JSON response
    response << "{\n";
    response << "  \"cross_solution\":[\n";
    response << "    \"";
    for (int move : crossSol) {
        response << ACTIONS.at(move) << " ";
    }
    response << "\"\n";
    response << "  ],\n";

    // Add all F2L solutions to the JSON response
    int counter = 0;
    for (const auto &f2l_sol : allSol) {
        response << "  \"solution\" << counter++ << "\":[\n";
        for (size_t i = 0; i < f2l_sol.size(); ++i) {
            response << "    \"";
            for (int move : f2l_sol[i]) {
                response << ACTIONS.at(move) << " ";
            }
            response << "\"";
            if (i != f2l_sol.size() - 1) {
                response << ",";
            }
            response << "\n";
        }
        response << "  ]";
        if (counter != allSol.size()) {
            response << ",";
        }
    }
}

```



```

        response << "\n";
    }

    response << "}\n";
    return response.str();
}

int main(int argc, const char *argv[]) {
    served::multiplexer mux;

    // Handle /api/v1/greeting endpoint
    mux.handle("/api/v1/greeting")
        .get([](served::response &res, const served::request &req) {
            // Extract scramble query parameter
            std::string scramble = req.query["scramble"];

            // Process each request asynchronously
            auto future_result = std::async(std::launch::async,
process_solution, scramble);
            std::string result = future_result.get();

            // Set response headers and body
            res.set_header("content-type", "application/json");
            res << result;
        });

    // Handle /api/v1/greeting_one_by_one endpoint
    mux.handle("/api/v1/greeting_one_by_one")
        .get([](served::response &res, const served::request &req) {
            // Extract scramble and order query parameters
            std::string scramble = req.query["scramble"];
            std::string order_str = req.query["order"];

            // Convert order string to vector of integers
            std::vector<int> order;
            std::istringstream iss(order_str);
            std::string token;
            while (std::getline(iss, token, ',')) {
                order.push_back(std::stoi(token));
            }

            // Process each request asynchronously
            auto future_result = std::async(std::launch::async,
process_solution_one_by_one, scramble, order);
            std::string result = future_result.get();

            // Set response headers and body
            res.set_header("content-type", "application/json");
            res << result;
        });
}

```

```

    // Print example curl commands for testing
    std::cout << "Try these examples with:" << std::endl;
    std::cout << "    curl \"http://localhost:8123/api/v1/greeting?scramble='R U R U'\"" << std::endl;
    std::cout << "    curl \"http://localhost:8123/api/v1/greeting_one_by_one?scramble='R U R U'&order=1,2,3,4\"" << std::endl;

    // Run server with multiple threads (10 threads)
    served::net::server server("127.0.0.1", "8123", mux);
    server.run(10); // 10 threads for handling requests

    return (EXIT_SUCCESS);
}

```

## proxy.js

Šis kods veido Express serveri, kas apkalpo vairākus API maršrutus, izmantojot MongoDB datubāzi, lai pārvaldītu Rubika kuba OLL un PLL algoritmu datus un vēlamos algoritmus. Tiek izmantotas .env vides mainīgie, lai atdalītu sensitīvu informāciju (piemēram, MongoDB akreditācijas datus) no koda. API nodrošina funkcionalitāti, piemēram, algoritmu iegūšanu, lietotāju datu saglabāšanu un prioritāšu atjaunināšanu. Ir arī ieviesta pieprasījumu rinda, lai kontrolētu to izpildes secību.

```

// Load environment variables from .env file in the current directory
require('dotenv').config({ path: __dirname + '/.env' });

const express = require('express');
const cors = require('cors');
const axios = require('axios');
const { MongoClient, ServerApiVersion } = require('mongodb');

// Load MongoDB and other config values from environment variables
const MONGODB_USERNAME = process.env.MONGODB_USERNAME
const MONGODB_PASSWORD = process.env.MONGODB_PASSWORD
const MONGODB_CLUSTER = process.env.MONGODB_CLUSTER
const MONGODB_DATABASE = process.env.MONGODB_DATABASE
const MONGODB_APPNAME = process.env.MONGODB_APPNAME

const PROJECTID = process.env.PROJECTID;

const LOCAL_API_ENDPOINT = process.env.LOCAL_API_ENDPOINT
const SERVER_PORT = process.env.SERVER_PORT

// Helper to build the MongoDB connection URI
const getMongoURI = () => {
    return
`mongodb+srv://${MONGODB_USERNAME}:${encodeURIComponent(MONGODB_PASSWORD)}@${M
ONGODB_CLUSTER}/?retryWrites=true&w=majority&appName=${MONGODB_APPNAME}`;

```

```

};

const app = express();

// Enable CORS for all origins and allow GET/POST methods
app.use(cors({
  origin: '*',
  methods: ['GET', 'POST'],
}));
app.options('*', cors());
app.use(express.json());

// Simple request queue to process tasks one by one
let requestQueue = [];
let processing = false;

// Add a task to the queue and start processing if not already running
function addToQueue(task) {
  requestQueue.push(task);
  processQueue();
}

// Process the queue sequentially
async function processQueue() {
  if (processing || requestQueue.length === 0) return;

  processing = true;
  const task = requestQueue.shift();

  try {
    await task();
  } catch (e) {
    console.error('Error while processing task:', e);
  } finally {
    processing = false;
    processQueue();
  }
}

// Proxy endpoint: forwards request to local API, processes one at a time
app.get('/api/proxy/oneByOne', (req, res) => {
  const scramble = req.query.scramble;
  const order = req.query.order;

  if (!scramble) {
    return res.status(400).json({ error: 'Scramble parameter is required' });
  }

  addToQueue(async () => {
    try {

```

```

        const response = await
axios.get(`${LOCAL_API_ENDPOINT}/api/v1/greeting_one_by_one?scramble=${encodeURIComponent(scramble)}&order=${encodeURIComponent(order)}`);
        res.json(response.data);
    } catch (error) {
        res.status(500).json({ error: 'Failed to fetch data'
    });
    }
});

// Returns the project ID from environment variables
app.get('/api/proxy/getProjectId', (req, res) => {
    if (!PROJECTID) {
        return res.status(500).json({ error: 'Project ID is not set'
    });
    }
    res.json({ projectId: PROJECTID });
});

// Fetch all OLL cases from MongoDB
app.get('/api/proxy/oll/all-cases', async (req, res) => {
    const uri = getMongoURI();
    const client = new MongoClient(uri, {
        serverApi: {
            version: ServerApiVersion.v1,
            strict: true,
            deprecationErrors: true,
        }
    });

    try {
        await client.connect();
        const collection =
client.db(MONGODB_DATABASE).collection("oll_index");
        const results = await collection.find({}).toArray();
        res.json(results);
    } catch (error) {
        console.error("MongoDB error:", error);
        res.status(500).json({ error: "Internal Server Error" });
    } finally {
        await client.close();
    }
});

// Fetch all PLL cases from MongoDB
app.get('/api/proxy/pll/all-cases', async (req, res) => {
    const uri = getMongoURI();
    const client = new MongoClient(uri, {
        serverApi: {
            version: ServerApiVersion.v1,

```

```

        strict: true,
        deprecationErrors: true,
    }
});

    try {
        await client.connect();
        const collection =
client.db(MONGODB_DATABASE).collection("pll_index");
        const results = await collection.find({}).toArray();
        res.json(results);
    } catch (error) {
        console.error("MongoDB error:", error);
        res.status(500).json({ error: "Internal Server Error" });
    } finally {
        await client.close();
    }
});

// Fetch all OLL algorithms from MongoDB
app.get('/api/proxy/oll/all-algs', async (req, res) => {
    const uri = getMongoURI();

    const client = new MongoClient(uri, {
        serverApi: {
            version: ServerApiVersion.v1,
            strict: true,
            deprecationErrors: true,
        }
    });

    try {
        await client.connect();
        const collection =
client.db(MONGODB_DATABASE).collection("oll_algs");
        const results = await collection.find({}).toArray();
        res.json(results);
    } catch (error) {
        console.error("MongoDB error:", error);
        res.status(500).json({ error: "Internal Server Error" });
    } finally {
        await client.close();
    }
});

// Fetch all PLL algorithms from MongoDB
app.get('/api/proxy/pll/all-algs', async (req, res) => {
    const uri = getMongoURI();
    const client = new MongoClient(uri, {
        serverApi: {
            version: ServerApiVersion.v1,

```

```

        strict: true,
        deprecationErrors: true,
    }
});

try {
    await client.connect();
    const collection =
client.db(MONGODB_DATABASE).collection("pll_algs");
    const results = await collection.find({}).toArray();
    res.json(results);
} catch (error) {
    console.error("MongoDB error:", error);
    res.status(500).json({ error: "Internal Server Error" });
} finally {
    await client.close();
}
});

// Add a new user to the database
app.post('/api/proxy/addUserDB', async (req, res) => {
    const { userId } = req.body;

    if (!userId) {
        return res.status(400).json({ error: "UserId is required" });
    }

    const uri = getMongoURI();
    const client = new MongoClient(uri, {
        serverApi: {
            version: ServerApiVersion.v1,
            strict: true,
            deprecationErrors: true,
        }
    });

    try {
        await client.connect();
        const collection =
client.db(MONGODB_DATABASE).collection("users");

        const result = await collection.insertOne({ userId });
        res.status(201).json({ message: "User added successfully",
result });
    } catch (error) {
        console.error("MongoDB error:", error);
        res.status(500).json({ error: "Internal Server Error" });
    } finally {
        await client.close();
    }
});
});

```

```

// Add a preferred OLL algorithm for a user
app.post('/api/proxy/addUserPrefOLL', async (req, res) => {
    const { userId, ollId, scramble, priority } = req.body;

    if (!userId || !ollId || !scramble || priority === undefined) {
        return res.status(400).json({ error: "UserId, ollId, scramble,
and priority are required" });
    }

    const uri = getMongoURI();
    const client = new MongoClient(uri, {
        serverApi: {
            version: ServerApiVersion.v1,
            strict: true,
            deprecationErrors: true,
        }
    });

    try {
        await client.connect();

        const prefOLLAlgsCollection =
client.db(MONGODB_DATABASE).collection("pref_oll_algs");

        // Check if the preference already exists
        const existingPreference = await
prefOLLAlgsCollection.findOne({
            userId,
            ollId,
            scramble,
        });

        if (existingPreference) {
            return res.status(400).json({ error: "Preference
already exists" });
        }

        const result = await prefOLLAlgsCollection.insertOne({
            userId,
            ollId,
            scramble,
            priority,
        });

        res.status(201).json({ message: "Preference added
successfully", result });
    } catch (error) {
        console.error("MongoDB error:", error);
        res.status(500).json({ error: "Internal Server Error" });
    } finally {

```

```

        await client.close();
    }
});

// Get all preferred OLL algorithms for a user
app.get('/api/proxy/getUserPrefOLL', async (req, res) => {
    const { userId } = req.query;

    if (!userId) {
        return res.status(400).json({ error: "UserId is required" });
    }

    const uri = getMongoURI();
    const client = new MongoClient(uri, {
        serverApi: {
            version: ServerApiVersion.v1,
            strict: true,
            deprecationErrors: true,
        }
    });

    try {
        await client.connect();

        const prefOLLAlgsCollection =
client.db(MONGODB_DATABASE).collection("pref_oll_algs");

        // Retrieve preferences for the user
        const preferences = await prefOLLAlgsCollection.find({ userId
}).toArray();

        res.status(200).json(preferences);
    } catch (error) {
        console.error("MongoDB error:", error);
        res.status(500).json({ error: "Internal Server Error" });
    } finally {
        await client.close();
    }
});

// Delete a preferred OLL algorithm for a user
app.post('/api/proxy/deleteUserPrefOLL', async (req, res) => {
    const { userId, ollId, scramble } = req.body;

    if (!userId || !ollId || !scramble) {
        return res.status(400).json({ error: "UserId, ollId, and
scramble are required" });
    }

    const uri = getMongoURI();
    const client = new MongoClient(uri, {

```



```

        serverApi: {
            version: ServerApiVersion.v1,
            strict: true,
            deprecationErrors: true,
        }
    });

    try {
        await client.connect();

        const prefOLLAlgsCollection =
client.db(MONGODB_DATABASE).collection("pref_oll_algs");

        // Delete the preference from the pref_oll_algs collection
        const result = await prefOLLAlgsCollection.deleteOne({
            userId,
            ollId,
            scramble,
        });

        if (result.deletedCount === 0) {
            return res.status(404).json({ error: "Preference not
found" });
        }

        res.status(200).json({ message: "Preference deleted
successfully" });
    } catch (error) {
        console.error("MongoDB error:", error);
        res.status(500).json({ error: "Internal Server Error" });
    } finally {
        await client.close();
    }
});

// Update the priority of a preferred OLL algorithm for a user
app.post('/api/proxy/updatePriority', async (req, res) => {
    const { userId, ollId, scramble, priority } = req.body;

    if (!userId || !ollId || !scramble || priority === undefined) {
        return res.status(400).json({ error: "UserId, ollId, scramble,
and priority are required" });
    }

    const uri = getMongoURI();
    const client = new MongoClient(uri, {
        serverApi: {
            version: ServerApiVersion.v1,
            strict: true,
            deprecationErrors: true,
        }
    });

```

```

    });

    try {
        await client.connect();

        const prefOLLAlgsCollection =
client.db(MONGODB_DATABASE).collection("pref_oll_algs");

        // Update the priority of the preference
        const result = await prefOLLAlgsCollection.updateOne(
            { userId, ollId, scramble },
            { $set: { priority } }
        );

        if (result.matchedCount === 0) {
            return res.status(404).json({ error: "Preference not
found" });
        }

        res.status(200).json({ message: "Priority updated
successfully" });
    } catch (error) {
        console.error("MongoDB error:", error);
        res.status(500).json({ error: "Internal Server Error" });
    } finally {
        await client.close();
    }
});

// Add a preferred PLL algorithm for a user
app.post('/api/proxy/addUserPrefPLL', async (req, res) => {
    const { userId, pllId, scramble, priority } = req.body;

    if (!userId || !pllId || !scramble || priority === undefined) {
        return res.status(400).json({ error: "UserId, pllId, scramble,
and priority are required" });
    }

    const uri = getMongoURI();
    const client = new MongoClient(uri, {
        serverApi: {
            version: ServerApiVersion.v1,
            strict: true,
            deprecationErrors: true,
        }
    });

    try {
        await client.connect();

```

```

        const prefPLLAlgsCollection =
client.db(MONGODB_DATABASE).collection("pref_pll_algs");

        // Check if the preference already exists
        const existingPreference = await
prefPLLAlgsCollection.findOne({
            userId,
            pllId,
            scramble,
        });

        if (existingPreference) {
            return res.status(400).json({ error: "Preference
already exists" });
        }

        const result = await prefPLLAlgsCollection.insertOne({
            userId,
            pllId,
            scramble,
            priority,
        });

        res.status(201).json({ message: "Preference added
successfully", result });
    } catch (error) {
        console.error("MongoDB error:", error);
        res.status(500).json({ error: "Internal Server Error" });
    } finally {
        await client.close();
    }
});

// Get all preferred PLL algorithms for a user
app.get('/api/proxy/getUserPrefPLL', async (req, res) => {
    const { userId } = req.query;

    if (!userId) {
        return res.status(400).json({ error: "UserId is required" });
    }

    const uri = getMongoURI();
    const client = new MongoClient(uri, {
        serverApi: {
            version: ServerApiVersion.v1,
            strict: true,
            deprecationErrors: true,
        }
    });

    try {

```

```

        await client.connect();

        const prefPLLAigsCollection =
client.db(MONGODB_DATABASE).collection("pref_pll_algs");

        // Retrieve preferences for the user
        const preferences = await prefPLLAigsCollection.find({ userId
}).toArray();

        res.status(200).json(preferences);
    } catch (error) {
        console.error("MongoDB error:", error);
        res.status(500).json({ error: "Internal Server Error" });
    } finally {
        await client.close();
    }
});

// Delete a preferred PLL algorithm for a user
app.post('/api/proxy/deleteUserPrefPLL', async (req, res) => {
    const { userId, pllId, scramble } = req.body;

    if (!userId || !pllId || !scramble) {
        return res.status(400).json({ error: "UserId, pllId, and
scramble are required" });
    }

    const uri = getMongoURI();
    const client = new MongoClient(uri, {
        serverApi: {
            version: ServerApiVersion.v1,
            strict: true,
            deprecationErrors: true,
        }
    });

    try {
        await client.connect();

        const prefPLLAigsCollection =
client.db(MONGODB_DATABASE).collection("pref_pll_algs");

        // Delete the preference from the pref_pll_algs collection
        const result = await prefPLLAigsCollection.deleteOne({
            userId,
            pllId,
            scramble,
        });

        if (result.deletedCount === 0) {

```

```

        return res.status(404).json({ error: "Preference not
found" });
    }

    res.status(200).json({ message: "Preference deleted
successfully" });
} catch (error) {
    console.error("MongoDB error:", error);
    res.status(500).json({ error: "Internal Server Error" });
} finally {
    await client.close();
}
});

// Update the priority of a preferred PLL algorithm for a user
app.post('/api/proxy/updatePriorityPLL', async (req, res) => {
    const { userId, pllId, scramble, priority } = req.body;

    if (!userId || !pllId || !scramble || priority === undefined) {
        return res.status(400).json({ error: "UserId, pllId, scramble,
and priority are required" });
    }

    const uri = getMongoURI();
    const client = new MongoClient(uri, {
        serverApi: {
            version: ServerApiVersion.v1,
            strict: true,
            deprecationErrors: true,
        }
    });

    try {
        await client.connect();

        const prefPLLAlgsCollection =
client.db(MONGODB_DATABASE).collection("pref_pll_algs");

        // Update the priority of the preference
        const result = await prefPLLAlgsCollection.updateOne(
            { userId, pllId, scramble },
            { $set: { priority } }
        );

        if (result.matchedCount === 0) {
            return res.status(404).json({ error: "Preference not
found" });
        }

        res.status(200).json({ message: "Priority updated
successfully" });
    }

```

```

    } catch (error) {
      console.error("MongoDB error:", error);
      res.status(500).json({ error: "Internal Server Error" });
    } finally {
      await client.close();
    }
  });

// Start the Express server
app.listen(SERVER_PORT, () => console.log(`Proxy server running on port ${SERVER_PORT}`));

```

## getSol.js

Šis JavaScript kods pievieno galveno meklēšanas funkciju kas startē verifikāciju: izlasot ievadīto scramble, to nosūtot uz backend /api/proxy/oneByOne, lai iegūtu krusta un F2L risinājumus pa pāriem. Tad tas veic virtuālā kuba simulāciju, nosaka OLL un PLL gadījumus, pārbauda lietotāja preferētos algoritmus vai noklusējuma variantus un mēģina atrisināt. Vairākas permūtācijas un papildus pārvietojumi tiek testēti secīgi ar progresu atjaunošanu un risinājumu filtrēšanu. Beigās atrod visus risinājumus.

```

// Add click event listener to the "Get Solution" button
document.getElementById("get-sol").addEventListener("click", async function ()
{
  this.disabled = true; // Disable button to prevent multiple clicks

  const scramble = document.getElementById("scramble").value;
  console.log("Scramble:", scramble);
  solutions = [];
  currentStepCount = 0;
  solvingInProgress = true;

  cleanupFilterModals(); // Remove any open filter modals

  createLoadingUI(); // Show loading UI

  // Clear any previous solving timeouts/intervals
  if (window.solvingTimeout) {
    clearTimeout(window.solvingTimeout);
  }
  if (window.solvingInterval) {
    clearInterval(window.solvingInterval);
  }

  // Main function to find solutions for a given scramble, permutation, and
  extra moves
  async function findSolutions(scramble, perm, extraMoves){

```

```

const scrambleORG = scramble;
scramble+= " " + extraMoves
cube = new Cube(3);
const url =
`http://localhost:3000/api/proxy/oneByOne?scramble=${encodeURIComponent(scramble)}&order=${encodeURIComponent(perm.join(","))}`;
console.log("URL:", url);

try {
  // Fetch cross and F2L solution from backend
  const response = await fetch(url, { method: "GET", headers: {
    "Content-Type": "application/json" } });
  if (!response.ok) throw new Error("Failed to fetch oneByOne solution");

  const data = await response.json();
  console.log("OneByOne Data:", data);

  // Apply scramble, cross, F2L, and z2 rotation to cube
  cube.doAlgorithm(scramble);
  cube.doAlgorithm(data.cross_solution.join(" "));
  data.solution0.forEach(step => cube.doAlgorithm(step));
  cube.doAlgorithm("z2");

  let cubeRepresentation = [cube.cube];

  // Find OLL case by matching cube state
  let ollCase = function () {
    for (let i = 0; i < ollIndex.length; i++) {
      if (JSON.stringify(ollIndex[i].cube_representation) ===
JSON.stringify(cubeRepresentation)) {
        return ollIndex[i];
      }
    }
    return null;
  }

  let howManyUs = 0;
  // Try up to 4 U moves to match OLL case
  while (ollCase() === null && howManyUs < 4) {
    cube.doAlgorithm("U");
    howManyUs++;
    cubeRepresentation = [cube.cube];
  }
  let ollCaseId = -1;
  if(ollCase() !== null) {
    ollCaseId = parseInt((ollCase().oll_id).replace("OLL ", ""),
10);
  }

  console.log("OLL Case ID:", ollCaseId);

```

```

// Get OLL algorithms based on user preferences
let ollAlgs = function () {
  if (ollCaseId === -1) {
    return ["U U"];
  }

  const ollIdString = `OLL ${ollCaseId}`;
  const userPrefs = userOllPrefs.filter(pref => pref.ollId ===
ollIdString);

  if (useUserPrefs === 0) {
    // Use default algorithms only
    console.log(`Using default algorithms for
${ollIdString}`);
    for (let i = 0; i < ollTable.length; i++) {
      if (ollTable[i].oll_id === ollCaseId - 1) {
        return ollTable[i].algorithms;
      }
    }
    return null;
  }

  if (useUserPrefs === 1) {
    // Prioritize user algorithms, then default
    if (userPrefs.length > 0) {
      userPrefs.sort((a, b) => a.priority - b.priority);

      let defaultAlgs = [];
      for (let i = 0; i < ollTable.length; i++) {
        if (ollTable[i].oll_id === ollCaseId - 1) {
          defaultAlgs = ollTable[i].algorithms;
          break;
        }
      }

      const userAlgs = userPrefs.map(pref => pref.scramble);
      console.log(`Prioritizing user's algorithms for
${ollIdString}:`, userAlgs);

      return [...userAlgs, ...defaultAlgs.filter(alg =>
!userAlgs.includes(alg))];
    }

    // No user preferences, use default
    console.log(`No user preferences for ${ollIdString}, using
default algorithms`);
    for (let i = 0; i < ollTable.length; i++) {
      if (ollTable[i].oll_id === ollCaseId - 1) {
        return ollTable[i].algorithms;
      }
    }
  }
}

```



```

    }
    return null;
}

if (useUserPrefs === 2) {
    // Use only user algorithms, fallback to default if none
    if (userPrefs.length > 0) {
        userPrefs.sort((a, b) => a.priority - b.priority);

        const userAlgs = userPrefs.map(pref => pref.scramble);
        console.log(`Using only user's algorithms for
${ollIdString}:`, userAlgs);
        return userAlgs;
    }

    // No user preferences, fallback to default
    console.log(`No user preferences for ${ollIdString},
falling back to default algorithms`);
    for (let i = 0; i < ollTable.length; i++) {
        if (ollTable[i].oll_id === ollCaseId - 1) {
            return ollTable[i].algorithms;
        }
    }
    return null;
}

return null;
}

let ollAlgsTable = ollAlgs();

if (ollAlgsTable === null) {
    ollAlgsTable = [];
    ollAlgsTable.push("U U'");
}

// Try up to 5 OLL algorithms for this case
for (const ollAlg of ollAlgsTable.slice(0, 5)) {
    let pllCube = new Cube(4);
    pllCube.doAlgorithm(scramble);
    pllCube.doAlgorithm(data.cross_solution.join(" "));
    for(const step of data.solution0) {
        pllCube.doAlgorithm(step);
    }
    pllCube.doAlgorithm("z2");
    for (let i = 0; i < howManyUs; i++) {
        pllCube.doAlgorithm("U");
    }
    pllCube.doAlgorithm(ollAlg);
    let pllCubeRepresentation = pllCube.cube;

    // Find PLL case by matching cube state

```

```

        let pllCase = function () {
            for (let i = 0; i < pllIndex.length; i++) {
                for (let j = 0; j <
pllIndex[i].cube_representation.length; j++) {
                    if
(JSON.stringify(pllIndex[i].cube_representation[j]) ===
JSON.stringify(pllCubeRepresentation)) {
                        return pllIndex[i];
                    }
                }
            }
            return null;
        }

let howManyUsPLL = 0;
let count = 0;
// Try up to 4 U moves to match PLL case
while (pllCase() === null && count < 4) {
    count++;
    pllCube.doAlgorithm("U");
    howManyUsPLL++;
    pllCubeRepresentation = pllCube.cube;
}
let pllCaseId = 0;
if(pllCase() !== null){
    pllCaseId = pllCase().pll_id;
}

console.log("PLL Case ID:", pllCaseId);

// Get PLL algorithms based on user preferences
let pllAlgs = function () {
    if (pllCaseId === 0) {
        return null;
    }

    const userPrefs = userPllPrefs.filter(pref => pref.pllId
=== pllCaseId);

    if (useUserPrefs === 0) {
        // Use default algorithms only
        console.log(`Using default algorithms for PLL
${pllCaseId}`);

        for (let i = 0; i < pllTable.length; i++) {
            if (pllTable[i].pll_id === pllCaseId) {
                return pllTable[i].algorithms;
            }
        }
        return null;
    }
}

```

```

    if (useUserPrefs === 1) {
      // Prioritize user algorithms, then default
      if (userPrefs.length > 0) {
        userPrefs.sort((a, b) => a.priority - b.priority);

        let defaultAlgs = [];
        for (let i = 0; i < pllTable.length; i++) {
          if (pllTable[i].pll_id === pllCaseId) {
            defaultAlgs = pllTable[i].algorithms;
            break;
          }
        }

        const userAlgs = userPrefs.map(pref =>
pref.scramble);

        console.log(`Prioritizing user's algorithms for
PLL ${pllCaseId}:`, userAlgs);
        return [...userAlgs, ...defaultAlgs.filter(alg =>
!userAlgs.includes(alg))];
      }

      // No user preferences, use default
      console.log(`No user preferences for PLL ${pllCaseId},
using default algorithms`);
      for (let i = 0; i < pllTable.length; i++) {
        if (pllTable[i].pll_id === pllCaseId) {
          return pllTable[i].algorithms;
        }
      }
      return null;
    }

    if (useUserPrefs === 2) {
      // Use only user algorithms, fallback to default if
none

      if (userPrefs.length > 0) {
        userPrefs.sort((a, b) => a.priority - b.priority);

        const userAlgs = userPrefs.map(pref =>
pref.scramble);

        console.log(`Using only user's algorithms for PLL
${pllCaseId}:`, userAlgs);
        return userAlgs;
      }

      // No user preferences, fallback to default
      console.log(`No user preferences for PLL ${pllCaseId},
falling back to default algorithms`);
      for (let i = 0; i < pllTable.length; i++) {
        if (pllTable[i].pll_id === pllCaseId) {
          return pllTable[i].algorithms;

```

```

        }
    }
    return null;
}

return null;
}
let pllAlgsTable = pllAlgs();

// If no PLL algorithms found, handle skipped PLL
if (pllAlgsTable === null) {
    if(ollAlg === "U U'") {
        // Handle full skip (no OLL/PLL)
        const finalAUFcube = new Cube(0);
        finalAUFcube.doAlgorithm(scramble);
        finalAUFcube.doAlgorithm(data.cross_solution.join(
""));

        data.solution0.forEach(step =>
finalAUFcube.doAlgorithm(step));
        finalAUFcube.doAlgorithm("z2");
        for (let i = 0; i < howManyUs; i++) {
            finalAUFcube.doAlgorithm("U");
        }

        let finalAUF = "";
        if (!finalAUFcube.isSolved()) {
            // Try all possible AUF moves to see if cube can
be solved

            const possibleMoves = [
                "R", "R'", "R2",
                "L", "L'", "L2",
                "U", "U'", "U2",
                "D", "D'", "D2",
                "F", "F'", "F2",
                "B", "B'", "B2"
            ];

            let isSolved = false;
            for (const move of possibleMoves) {
                const testCube = new Cube(0);
                testCube.doAlgorithm(scramble);
                testCube.doAlgorithm(data.cross_solution.join(
" "));

                data.solution0.forEach(step =>
testCube.doAlgorithm(step));

                testCube.doAlgorithm("z2");
                for (let i = 0; i < howManyUs; i++) {
                    testCube.doAlgorithm("U");
                }
                testCube.doAlgorithm(move);

```

```

        if (testCube.isSolved()) {
            finalAUF = move;
            isSolved = true;
            break;
        }
    }
}

// Add solution with skipped OLL/PLL
addSolution({
    scramble: scrambleORG,
    crossSolution: extraMoves + " " +
data.cross_solution.join(" "),
    f2lSolution: data.solution0,
    ollAlgorithm: "Skipped",
    pllAlgorithm: "Skipped" + (finalAUF ? " " +
finalAUF : ""),

    howManyUs: howManyUs,
    howManyUsPLL: howManyUsPLL,
    ollCaseId: null,
    pllCaseId: null
});
}

// Handle skipped PLL only
const finalAUFCube = new Cube(0);
finalAUFCube.doAlgorithm(scramble);
finalAUFCube.doAlgorithm(data.cross_solution.join(" "));
data.solution0.forEach(step =>
finalAUFCube.doAlgorithm(step));
finalAUFCube.doAlgorithm("z2");
for (let i = 0; i < howManyUs; i++) {
    finalAUFCube.doAlgorithm("U");
}
finalAUFCube.doAlgorithm(ollAlg);

let finalAUF = "";
if (!finalAUFCube.isSolved()) {
    // Try all possible AUF moves to see if cube can be
solved

    const possibleMoves = [
        "R", "R'", "R2",
        "L", "L'", "L2",
        "U", "U'", "U2",
        "D", "D'", "D2",
        "F", "F'", "F2",
        "B", "B'", "B2"
    ];

    let isSolved = false;
    for (const move of possibleMoves) {

```

```

        const testCube = new Cube(0);
        testCube.doAlgorithm(scramble);
        testCube.doAlgorithm(data.cross_solution.join("
    ));

        data.solution0.forEach(step =>
testCube.doAlgorithm(step));
        testCube.doAlgorithm("z2");
        for (let i = 0; i < howManyUs; i++) {
            testCube.doAlgorithm("U");
        }
        testCube.doAlgorithm(ollAlg);
        testCube.doAlgorithm(move);

        if (testCube.isSolved()) {
            finalAUF = move;
            isSolved = true;
            break;
        }
    }
}

// Add solution with skipped PLL
addSolution({
    scramble: scrambleORG,
    crossSolution: extraMoves + " " +
data.cross_solution.join(" "),
    f2lSolution: data.solution0,
    ollAlgorithm: ollAlg,
    pllAlgorithm: "Skipped" + (finalAUF ? " " + finalAUF :
    ""),

    howManyUs: howManyUs,
    howManyUsPLL: howManyUsPLL,
    ollCaseId: ollCaseId,
    pllCaseId: null
});
continue;
}

// Try up to 5 PLL algorithms for this case
for (const pllAlg of pllAlgsTable.slice(0, 5)) {
    const finalCube = new Cube(0);
    finalCube.doAlgorithm(scramble);
    finalCube.doAlgorithm(data.cross_solution.join(" "));
    data.solution0.forEach(step =>
finalCube.doAlgorithm(step));
    finalCube.doAlgorithm("z2");
    for (let i = 0; i < howManyUs; i++) {
        finalCube.doAlgorithm("U");
    }
    finalCube.doAlgorithm(ollAlg);
    for (let i = 0; i < howManyUsPLL; i++) {

```

```

        finalCube.doAlgorithm("U");
    }
    finalCube.doAlgorithm(p11Alg);

    let finalAUF = "";
    if (!finalCube.isSolved()) {
        // Try all possible AUF moves to see if cube can be
solved
        const possibleMoves = [
            "R", "R'", "R2",
            "L", "L'", "L2",
            "U", "U'", "U2",
            "D", "D'", "D2",
            "F", "F'", "F2",
            "B", "B'", "B2"
        ];

        let isSolved = false;
        for (const move of possibleMoves) {
            const testCube = new Cube(0);
            testCube.doAlgorithm(scramble);
            testCube.doAlgorithm(data.cross_solution.join("
"));

            data.solution0.forEach(step =>
testCube.doAlgorithm(step));
            testCube.doAlgorithm("z2");
            for (let i = 0; i < howManyUs; i++) {
                testCube.doAlgorithm("U");
            }
            testCube.doAlgorithm(ollAlg);
            for (let i = 0; i < howManyUsPLL; i++) {
                testCube.doAlgorithm("U");
            }
            testCube.doAlgorithm(p11Alg);
            testCube.doAlgorithm(move);

            if (testCube.isSolved()) {
                finalAUF = move;
                isSolved = true;
                break;
            }
        }
    }

    // Add solution, handle skipped OLL if needed
    if(ollAlg === "U U'") {
        addSolution({
            scramble: scrambleORG,
            crossSolution: extraMoves + " " +
data.cross_solution.join(" "),
            f2lSolution: data.solution0,

```

```

        ollAlgorithm: "Skipped",
        pllAlgorithm: pllAlg + (finalAUF ? " " + finalAUF
: ""),

        howManyUs: howManyUs,
        howManyUsPLL: howManyUsPLL,
        ollCaseId: null,
        pllCaseId: pllCaseId
    });
}
addSolution({
    scramble: scrambleORG,
    crossSolution: extraMoves + " " +
data.cross_solution.join(" "),
    f2lSolution: data.solution0,
    ollAlgorithm: ollAlg,
    pllAlgorithm: pllAlg + (finalAUF ? " " + finalAUF :
""),

    howManyUs: howManyUs,
    howManyUsPLL: howManyUsPLL,
    ollCaseId: ollCaseId,
    pllCaseId: pllCaseId
});
}
}
} catch (error) {
    console.error("Error solving cube:", error);
}
}
// Generate all permutations of F2L pairs
const permutations = generatePermutations([0, 1, 2, 3]);
let counter = 0;

totalSolveSteps = permutations.length * moves.length;

// Check if user is currently viewing a solution
const isViewingSolution = document.querySelector('.solution-container')
!== null;

// Loop through all permutations and moves
for (const perm of permutations) {
    if (!solvingInProgress) {
        console.log("Solving cancelled");
        break;
    }

    for (const move of moves) {
        if (!solvingInProgress) {
            console.log("Solving cancelled");
            break;
        }
    }
}

```



```

        await findSolutions(scramble, perm, move);
        counter++;
        currentStepCount++;
        console.log("Counter:", counter);
        solutions = sortSolutions(solutions);
        console.log("Solutions:", solutions);

        updateProgress(currentStepCount);

        if (counter === 2) {
            console.log("Solutions so far:", solutions.length);
        }
    }

    if (!solvingInProgress) {
        console.log("Solving cancelled");
        break;
    }
}

solvingInProgress = false;

document.getElementById("get-sol").disabled = false;

// Show solutions or update progress bar
if (isViewingSolution && solutions.length > 0) {
    updateMiniProgress();
} else {
    showFilteredSolutionsWithoutRefresh();
}
});

```