Bakalárska práca

Technická dokumentácia

Filip Híreš
Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií
xhires@stuba.sk
Máj 2025

1. Úvod

Aplikácia OPTIMUS vznikla ako súčasť bakalárskej práce na tému spracovanie a vizualizácia optimalizačných algoritmov. Jej cieľom je poskytnúť prehľad o princípoch fungovania rôznych optimalizačných techník a tiež mať možnosť si vyskúšať, ako vplývajú vstupné parametre na výkon optimalizačného procesu.

V tomto dokumente sa zameriame na zdokumentovanie vývoja aplikácie. Zhrnieme si, ako prebiehal návrh systému, použité technológie, systémovú architektúru, štruktúru zdrojového kódu a ďalšie technické detaily dôležité pre údržbu a ďalší vývoj systému. Dokumentácia je určená pre vývojárov a technických používateľov, ktorí sa chcú oboznámiť so spôsobom, akým bol systém navrhnutý a implementovaný. Systém bol vyvíjaný s dôrazom na rozšíriteľnosť a používateľskú prístupnosť.

2. Návrh systému

Prvým krokom pri návrhu systému bolo zadefinovanie požiadaviek na systém. Vzhľadom na zadanie bakalárskej práce boli funkčne a nie-funkčné požiadavky definované nasledovne.

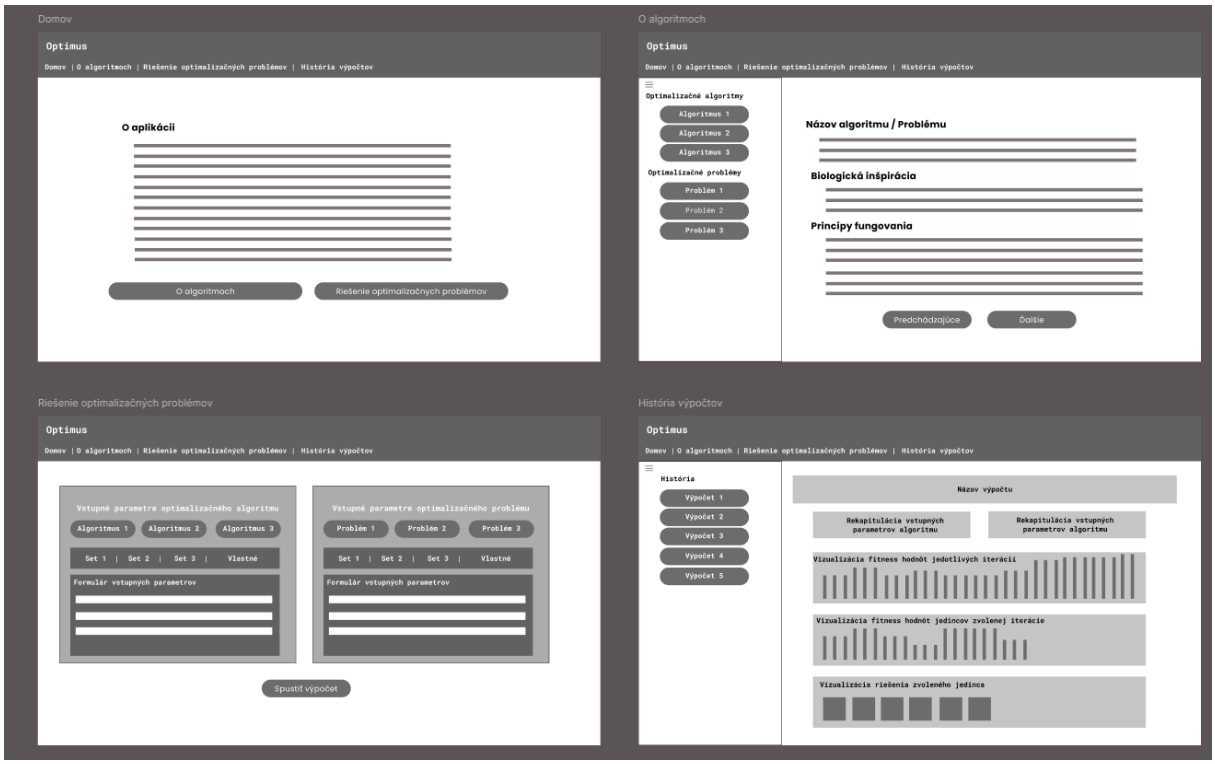
Funkčné požiadavky:

- Používateľ si bude môcť vybrať jeden z dostupných optimalizačných problémov
- Používateľ si bude môcť zvoliť algoritmus, ktorým sa bude problém riešiť
- Používateľ bude môcť ľubovoľne meniť parametre problémov a algoritmov alebo si vybrať z predefinovaných hodnôt
- Používateľ si bude môcť prezerať vizualizované výsledky algoritmov a štatistiky.
- Používateľ si bude môcť zobraziť informácie o jednotlivých optimalizačných algoritmoch a problémoch
- Systém bude ukladať výsledky výpočtov v rámci jednej relácie a používateľ si ich bude môcť opätovne prezerať

Nie-funkčné požiadavky:

- Systém bude navrhnutý tak, aby bolo jednoduché ho rozšíriť o ďalšie optimalizačné problémy a algoritmy
- Používateľské rozhranie bude intuitívne
- Aplikácia bude kompatibilná s bežne používanými internetovými prehliadačmi
- Optimalizačné algoritmy budú navrhnuté tak, aby ich výpočtová doba bola čo najkratšia
- Aplikácia bude responzívna, aby nestratila prehľadnosť na rôznych typoch zariadení

Následne boli vytvorené skice pre jednotlivé obrazovky používateľského rozhrania.



Na základe skíc a požiadaviek na systém bola následne zhotovená funkčná aplikácia. Kompletný návod pre prácu s používateľským rozhraním je uvedený v prílohe používateľská príručka.

3. Použité technológie

Implementácia bola zrealizovaná pomocou frameworkov Quasar a Vue.js. Vue.js je JavaScriptový framework, ktorý slúži na tvorbu reaktívnych systémov a správu dát. Má komponentovo založenú architektúru a umožňuje vývoj dynamických webových aplikácií, čo nám pomohlo pri zobrazovaní komponentov, ktoré boli v danej chvíli potrebné. Quasar obsahuje veľké množstvo predpripravených komponentov, ktoré sme použili v našej aplikácii.

Samotná logika aplikácie bola vytváraná pomocou jazyka TypeScript. Celý vývoj aplikácie sa odohrával v prostredí WebStorm od spoločnosti JetBrains.

Okrem týchto nástrojov sme použili aj niekoľko knižníc, ktoré výrazne uľahčili vývoj aplikácie. Konkrétne boli použité knižnice Pinia, Chart.js, a ECharts. Na uchovávanie dát bola použitá databáza prehliadača IndexedDB.

Pinia výrazne zjednodušila prenos dát medzi jednotlivými komponentami aplikácie pomocou mechanizmu Pinia stores. Pinia store predstavuje centralizované úložisko, ktoré uchováva stav aplikácie. Do tohto úložiska sa môžu dáta vložiť v jednom komponente aplikácie a následne k týmto dátam môže pristupovať ktorýkoľvek iný komponent v rámci aplikácie. V našej aplikácii je implementovaných niekoľko takýchto úložísk. Jedno z nich slúži na ukladanie a prístup k vstupným parametrom

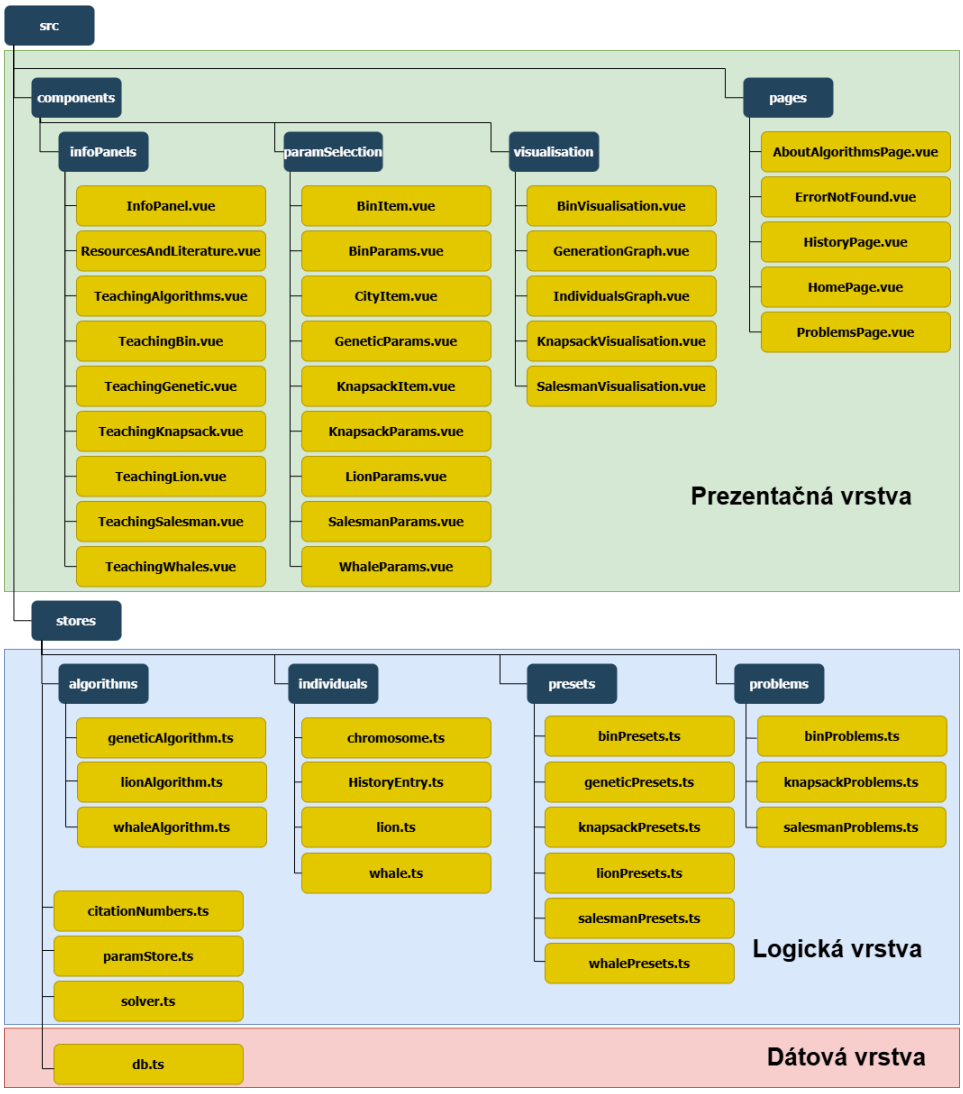
algoritmov. Toto úložisko zároveň zabezpečuje validáciu zadaných parametrov, čím sa zaručuje, že algoritmus bude spustený iba s korektnými a zmysluplnými údajmi. Ďalšie tri úložiská slúžia na uchovávanie funkcií pre konkrétne optimalizačné problémy.

Knižnice Chart.js a ECharts boli použité pri vizualizácii riešenia. Chart.js sme využili na vytvorenie a zobrazenie interaktívnych grafov, ktoré reprezentujú stav a vývoj fitness hodnôt jedincov. ECharts poslúžilo na reprezentáciu riešenia problému obchodného cestujúceho.

IndexedDB je databázový systém, ktorý umožňuje ukladať veľké množstvo dát priamo v prehliadači na strane klienta. Dáta sú z databázy zavolané len vtedy, keď je ich treba pre vizualizáciu.

4. Architektúra systému

Celý systém pozostáva z troch hlavných vrstiev. Prezentačná vrstva tvorí používateľské rozhranie. Logická vrstva riadi všetky výpočty, spracovanie dát a funkcionalitu systému. Na záver dátová vrstva ukladá všetky výstupné dáta aby ich bolo možné použiť neskôr. Na obrázku nižšie môžeme vidieť štruktúru aplikácie a zaradenie súborov do jednotlivých vrstiev.



Prezentačná vrstva – V rámci prezentačnej vrstvy existujú dva hlavné priečinky, ako je možné vidieť na predchádzajúcom obrázku. Priečinok pages obsahuje komponenty, ktoré udávajú rozloženie jednotlivých sekcií aplikácie (častí, do ktorých sa dá dostať cez navigačnú lištu). Priečinok components obsahuje ostatné komponenty, ktoré sa používajú v rámci jednotlivých sekcií. Komponenty sú podľa spôsobu ich využitia rozdelené na tri časti.

InfoPanels obsahuje komponenty využité v sekcii o algoritmoch, a teda v týchto komponentoch sa nachádzajú samotné informačné bloky. Pre každú záložku v knižnici existuje jeden príslušný komponent. ParamSelection obsahuje komponenty, ktoré slúžia na výber vstupných parametrov. V poslednom priečinku sa nachádzajú komponenty využité na vizualizáciu riešení a grafov.

Logická vrstva – Logická vrstva je celá uložená v priečinku stores. Nachádzajú sa tu súbory, ktoré sú zodpovedné za ukladanie vstupných parametrov a prístup k nim, manažment optimalizačného procesu a tiež niekoľko priečinkov ako algorithms a problems, ktoré obsahujú súbory s hlavnou logikou výpočtov. V priečinku individuals sa nachádzajú triedy reprezentujúce jedincov algoritmov a v priečinku presets sú uložené všetky sety predvolených parametrov.

Dátová vrstva – dátová vrstva je tvorená súborom db.ts, ktorý sa tiež nachádza v priečinku stores. V tomto súbore sa nachádzajú všetky funkcie pre vytvorenie a prácu s databázou.

5. Funkcionalita systému

V tejto kapitole si popíšeme najdôležitejšie funkcie v kóde. Ako prvé si rozoberieme funkcie zodpovedné za spravovanie databázy.

getDB()
Funkcia zabezpečuje otvorenie databázy optimizationAlgorithmDB s aktuálnou verziou. V prípade, že databáza ešte neexistuje, alebo sa zvyšuje jej verzia, funkcia vytvorí nové tabuľky.

getLatestVersion()
Pomocná funkcia, ktorá zisťuje aktuálnu verziu databázy. Otvorí existujúcu databázu a vráti jej verziu.

savePopulation()
Uloží jednu generáciu populácie do databázy.

getGenerationFromDB()
Načíta a vráti jednu konkrétnu generáciu z databázy na základe jej indexu.

getAllGenerationIndexes()
Z databázy získa zoznam všetkých indexov generácií, ktoré sú momentálne uložené.

clearGenerations()
Vymaže všetky generácie z databázy, čím sa resetuje obsah príslušnej tabuľky.

saveEntry()
Uloží jeden záznam typu HistoryEntry, ktorý obsahuje metadáta o konkrétnom behu algoritmu.

getEntryFromDB()
Z databázy načíta jeden konkrétny záznam typu HistoryEntry na základe jeho ID.

getAllEntryIndexes()
Získa všetky ID záznamov uložených v entries.

deleteEntry()
Vymaže jeden konkrétny záznam z entries podľa jeho ID.

resetEntriesObjectStore()
Komplexnejšia funkcia, ktorá vymaže a znovu vytvorí celý entries object store (s novou verziou databázy).

Keď už vieme, aké operácie nad databázou môžeme vykonať, môžeme si popísať priebeh optimalizácie. Celý proces začína stlačením tlačidla, ktoré zavolá funkciu checkInputs(). Táto funkcia overí vstupné parametre. Pokiaľ sú všetky korektne zadané, zavolá sa funkcia solve(), ktorá riadi celý proces.

solve()

Táto funkcia je dosť komplexná, a preto si zhrnieme jej najdôležitejšie časti. Najprv sa načíta pinia store, ktorý prislúcha zvolenému optimalizačnému problému a obsahuje všetky funkcie pre jeho riešenie. Následne sa na základe zvoleného optimalizačného algoritmu zavolá funkcia pre daný algoritmus. Funkcionalitu jednotlivých algoritmov si opíšeme neskôr. Dôležité však je, že pomocou funkcie savePopulation(), ktorá je spomenutá vyššie sa postupne do databázy uložia všetky populácie. Po dokončení výpočtov sa na základe optimalizačného problému opäť načítavajú populácie z databázy po jednej. Z každej populácie sú extrahované dáta potrebné pre vizualizáciu. Na konci sa všetky záznamy populácií vymažú z databázy, aby sa uvoľnil priestor pre ďalšie výpočty. Posledným krokom funkcie solve() je uloženie dát pre vizualizáciu v nasledujúcej podobe:

```
export class HistoryEntry {
 algorithm: string;
 problem: string;
 solution: number[][][][];
 fitness: number[][];
 bestFitness: number[];
 averageFitness: number[];
 mutation: number;
 elitism: boolean;
 elitismRate: number;
 choose: string;
 crossing: string;
 packs: number;
 females: number;
 males: number;
 hunters: number;
 capacity: number;
 averageWeight: number;
 averagePrice: number;
 count: number;
```

V takomto formáte je celý záznam uložený do databázy.

Genetický algoritmus – Populácia genetického algoritmu sa skladá z chromozómov. Trieda pre chromozóm vyzerá nasledovne:

```
export class Chromosome {
 solution: number[];
 fitness: number;

 constructor(solution: number[]) {
  this.solution = solution;
  this.fitness = 0;
 }
}
```

Solution je pole čísel, ktoré predstavuje riešenie problému a fitness je kvalitatívne ohodnotenie tohto riešenia.

Celý algoritmus prebieha nasledovne. Najprv sa vygeneruje počiatočná populácia a riešenia sa ohodnotia. Následne sa podľa počtu iterácií vyhodnocuje funkcia createNewGeneration(), ktorá prebieha v niekoľkých krokoch. Po vykonaní všetkých krokov v iterácii sa populácia uloží do databázy.

applyElitism()

Funkcia vyberá dané percento najlepších jedincov z populácie podľa hodnoty fitness a pridáva ich do novej generácie.

tournamentSelection()

Funkcia zabezpečuje turnajový výber. Z populácie náhodne vyberie podmnožinu jedincov a následne z nej vyberie najlepšieho jedinca s najvyššou hodnotou fitness.

rouletteSelection()

Funkcia vykonáva ruletový výber, kde je pravdepodobnosť výberu jedinca úmerná jeho fitness. Čím vyššia hodnota fitness, tým väčšia šanca na výber.

crossover()

Funkcia vytvára novú generáciu jedincov pomocou výberu rodičov, ich kríženia a aplikácie mutácie. Rodičia sú vyberaní pomocou turnaja alebo rulety, kríženie môže byť jednobodové, dvojbodové alebo uniformné, a následne sa aplikuje mutácia podľa nastavenej pravdepodobnosti.

evaluateIndividuals()

Funkcia vypočíta hodnotu fitness pre každého jedinca v populácii. Výpočet závisí od konkrétneho typu optimalizačného problému.

Algoritmus inšpirovaný správaním levov – Podobne ako v genetickom algoritme aj tu je každý jedinec reprezentovaný objektom triedy lion. Tá má nasledujúcu štruktúru:

```
export class Lion {
 solution: number[];
 fitness: number;
 sex: number;
 territory: number[];
 territoryValue: number;

 constructor(solution: number[], sex: number) {
  this.solution = solution;
  this.fitness = 0;
  this.sex = sex;
  this.territory = solution;
  this.territoryValue = 0;
 }
}
```

Teda okrem atribútov solution a fitness má každý objekt zadefinované pohlavie. Taktiež sú tu atribúty territory a territoryValue, ktoré fungujú rovnako ako riešenie a fitnes, ale označujú doteraz najlepšie nájdené riešenie daného jedinca. Algoritmus je zložený z nasledujúcich funkcií.

hunt()

Simuluje lovenie levíc v svorkách, rozdeľuje samice na loviace a neloviace, pričom loviace samice sa pokúšajú zamerať a priblížiť ku koristi, zatiaľ čo neloviace sa hýbu k lepším teritóriám. Samci zlepšujú svoje územia pohybom k najlepším riešeniam v rámci svorky. Nomádi skúmajú priestor náhodne.

breed()

Zabezpečuje kríženie samcov a samíc v rámci svoriek aj nomádov za účelom vytvorenia nových potomkov, pričom po krížení nasleduje aj mutácia.

migration()

Zabezpečuje migráciu jednotlivcov medzi svorkami a nomádmi a odstraňuje najslabšie riešenia.

Algoritmus inšpirovaný správaním veľrýb – Trieda, ktorá reprezentuje veľryby má rovnakú štruktúru ako chromozóm v genetickom algoritme:

```
export class Whale {
 solution: number[];
 fitness: number;

 constructor(solution: number[]) {
  this.solution = solution;
  this.fitness = 0;
 }
}
```

Okrem generických funkcií na výpočet fitness hodnôt a vygenerovanie počiatočnej populácie sa tu nachádza len jedna funkcia relocate(), ktorá zabezpečí aby sa všetky veľryby (okrem najlepšej) presunuli a tým preskúmali priestor riešení. Hlavná časť funkcie vyzerá takto:

```
population.forEach((individual) => {
 if (individual !== bestIndividual) {
  const p = Math.random()
  if (p < 0.5) {
   const a = 2 - iteration * (2 / maxIteration)
   const r = Math.random()
   const A = 2 * a * r - a
   let target = population[Math.floor(Math.random() * population.length)]
   if (A < 1) {
    target = bestIndividual;
   }
   // moving towards best or random solution - later iterations supports moving to
best while sooner are supporting exploration more
```

```
    if (target) {
      individual.solution = problemToSolve.move(individual.solution, target.solution,
0.3)
      newPopulation.push(individual)
    }
  }
  //or making spiral movement around target
  else {
    const target = bestIndividual
    if (target) {
      individual.solution = problemToSolve.spiralMove(target.solution)
      newPopulation.push(individual)
    }
  }
 } else {
  newPopulation.push(individual)
 }
})
```

Každý jedinec sa náhodne rozhodne, či sa bude pohybovať smerom k inému riešeniu (buď k najlepšiemu alebo náhodne vybranému), alebo sa bude pohybovať špirálovým spôsobom okolo najlepšieho riešenia. Toto rozhodnutie je náhodné, ale zohľadňuje sa číslo iterácie – na začiatku algoritmus viac skúma nové riešenia a neskôr sa viac zameriava na hľadanie okolo najlepších riešení.

Pohyb znamená, že jeho riešenie sa upraví tak, aby bolo podobnejšie riešeniu, ku ktorému je pohyb vykonaný. Ak sa rozhodne pre špirálový pohyb, riešenie sa pozmení v trajektórii okolo najlepšieho jedinca.

Najlepší jedinec v populácii sa nepohybuje len sa bez zmeny prenesie do novej generácie.

Každý optimalizačný problém obsahuje set funkcií, ktoré zabezpečujú, že každý algoritmus je použiteľný v kombinácii s každým problémom. Jedná sa o tieto funkcie.
createSolutions()

Vygeneruje náhodné počiatočné riešenia.

calculateFitness()

Vypočíta fitness hodnotu pre jedno konkrétne riešenie.

onePointCrossover()

Skombinuje riešenia dvoch jedincov tak, že v náhodnom bode sa ich riešenia rozdelia a vzájomne spoja. Vzniknú dve nové riešenia.

twoPointCrossover()

Funguje podobne ako predchádzajúca funkcia, ale používa dva náhodné body. Tým pádom sa vymení iba stredná časť riešenia medzi dvoma bodmi.

uniformCrossover()

Pri tomto spôsobe sa každá pozícia riešenia rozhoduje náhodne – buď si vezme hodnotu z prvého alebo druhého rodiča. Výsledkom sú opäť dve nové riešenia.

breed()

Používa sa v rámci algoritmu inšpirovaného levmi. Je to špecifická verzia kríženia, ktorá zaručuje, že jedno mláďa je samec a druhé samica.

move()

Upraví riešenie tak, že ho priblíži k cieľovému riešeniu. S určitou pravdepodobnosťou sa niektoré hodnoty vymenia za tie z cieľového riešenia.

spiralMove()

Simuluje „pohyby veľrýb“. Riešenie sa mierne zmutuje na základe toho, kde je cieľ – teda kde sa nachádza najlepšie riešenie.

mutate()

Pre každý jednotlivý chromozóm v populácii je šanca, že sa niektoré jeho hodnoty zmenia.

getProblemType()

Vracia názov aktuálne riešeného problému.

6. Inštalácia a spustenie

Na spustenie aplikácie je potrebné mať v systéme nainštalované základné vývojové nástroje, najmä Node.js a npm.

Inštalácia závislostí - V koreňovom adresári projektu sa spustí príkaz na inštaláciu všetkých potrebných balíkov a knižníc definovaných v súbore package.json. Tieto zahŕňajú napríklad knižnice Vue 3, Quasar, Pinia, Vue Router, a ďalšie závislosti potrebné pre beh aplikácie.

Spustenie vývojového servera

Po úspešnej inštalácii závislostí môže používateľ aplikáciu spustiť v režime pre vývoj, ktorý automaticky sleduje zmeny v kóde a automaticky obnovuje stránku. Aplikácia sa následne otvorí v predvolenom webovom prehliadači na lokálnej adrese.

7. Možné rozšírenia

Celá aplikácia je navrhnutá tak, že používateľské rozhranie je oddelené od logiky aplikácie. V budúcnosti bude teda možné rozšíriť aplikáciu o ďalšie prvky, bez zásahu do súčasnej implementácie.

Aplikáciu je možné rozšíriť o ďalšie optimalizačné algoritmy. Je možné pridať aj rôzne optimalizačné problémy pod podmienkou, že budú obsahovať potrebné funkcie, ktoré sú volané vo funkciách algoritmov ako napríklad funkcia na vytvorenie jedincov alebo vypočítanie hodnoty fitness. Aplikáciu bude tiež možné rozšíriť vytvorením ďalších vizualizačných komponentov.

8. Záver

Cieľom tejto technickej dokumentácie bolo podrobne popísať návrh, implementáciu a fungovanie aplikácie OPTIMUS. Dokumentácia poskytla prehľad návrhu a architektúry systému, použitých technológií a tiež popis kľúčových častí kódu. Systém bol navrhnutý s dôrazom na rozšíriteľnosť a intuitívne používateľské rozhranie.

Prípadné rozšírenia systému do budúcnosti môžu zahŕňať vytvorenie ďalších kombinácií algoritmov a riešených problémov a tiež väčšie možnosti vizualizácie výstupných dát. Veríme, že dokumentácia poslúži ako kvalitný základ pre ďalší vývoj, údržbu a prípadné nasadenie systému v reálnom prostredí.