OptiSlots

System Optymalizacji Ewolucyjnej Harmonogramów

Autors: Piotr Bonar · Jakub Borsuk · Aleksander Stepaniuk · Kacper Zakrzewski

Supervisor: dr hab. inż. Michał Przewoźniczek, Katedra Systemów i Sieci Komputerowych

Abstract

Obecne procesy tworzenia harmonogramów w organizacjach, takich jak uczelnie wyższe czy duże firmy, często opierają się na nieefektywnych i niesprawiedliwych metodach, jak np. model "kto pierwszy, ten lepszy". Powoduje to frustrację użytkowników, którzy muszą konkurować o miejsca w sztywno wyznaczonych oknach rejestracyjnych, generując przy tym ekstremalne obciążenia serwerów. Dla administratorów, np. dziekanatów, proces ten wiąże się z ogromnym nakładem pracy ręcznej, zarówno przy konfiguracji, jak i późniejszym korygowaniu licznych konfliktów i niespójności. Istniejące rozwiązania nie radzą sobie ze złożonością godzenia indywidualnych preferencji tysięcy użytkowników z twardymi ograniczeniami zasobów (np. dostępnością sal), co skutkuje niską satysfakcją wszystkich stron i planami, które często nie odpowiadają niczyim realnym potrzebom.

1 WPROWADZENIE

1.1 Cel projektu

Przedsięwzięcie miało na celu przyspieszyć i ułatwić proces ustalania harmonogramów w różnych jednostkach - od uczelni, przez firmy, aż po małe biznesy. Obecnie ustalanie harmonogramu to czasochłonny proces, w którym trzeba uwzględnić dostępność godzin, prowadzących, uczestników oraz miejsc spotkań.

Celem projektu było opracowanie systemu, który zautomatyzuje i zoptymalizuje proces tworzenia harmonogramów, co pozwoli na oszczędzanie czasu i zasobów oraz eliminację ryzyka konfliktów w planowaniu. Implementacja takiego rozwiązania przyniesie korzyści zarówno w kontekście biznesowym (efektywniejsze zarządzanie zasobami i lepsza organizacja pracy) jak i technicznym (łatwa rozbudowa i skalowalność).

(a) C++ (b) Django (c) Next.js (d) PostgreSQL

Figure 1: Użyte technologie

1.2 Powiązane projekty

Projekt można porównać z innymi narzędziami używanymi do tworzenia harmonogramów takich jak np.:

- USOS [13]
- ClickUp [3]
- Calendly [2]
- Kalendarz Google [6]

Zasadnicza różnica między podanymi produktami, a naszym systemem to fakt, że te serwisy służą do ręcznego tworzenia harmonogramów, gdzie firma musi przypisać spotkania do danego użytkownika. Nasz system automatycznie tworzy harmonogramy na podstawie podanych preferencji oraz parametrów ustalanych przez administrację firmy.

2 REZULTATY

W ramach projektu stworzono aplikację webową pozwalającą firmom tworzyć oraz zarządzać harmonogramami. System składa się z trzech głównych komponentów - frontendu, REST API, optymalizatora wykorzystującego algorytm genetyczny, oraz pomocniczych modułów:

- Baza danych PostgreSQL
- Scheduler - Przekazuje gotowe dane dla optymalizatora
- Progress listener - odbiera gotowe rozwiązania
- Redis - kolejka zadań dla optymalizatora

2.0.1 Frontend - Next.js [14]

Frontend napisany został w języku React i wykorzystuje framework Next.js. Obsługuje potrzeby użytkowników korzystających z systemu. Użytkownicy dodani do firmy przez administratora systemu lub przez innego użytkownika posiadającego odpowiednie uprawnienia zyskują dostęp do funkcjonalności zależnych od ich roli.

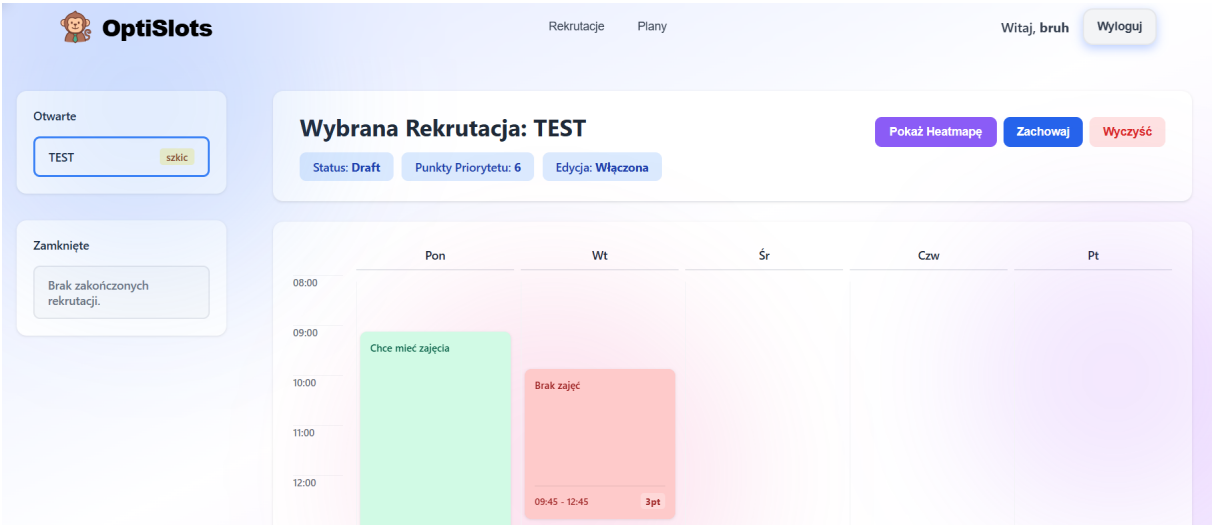

Figure 2: Widok frontendu użytkownika

Użytkownicy prowadzący spotkania (’hosts’) oraz uczestniczący w spotkaniach (’participants’) mogą dodawać, edytować i usuwać swoje preferencje [2]. Użytkownik ustala preferencje dla każdej z rekrutacji, a system dba aby użytkownik nie był w sytuacji, w której wygenerowane harmonogramy są ze sobą sprzeczne - na przykład wymagają obecności w dwóch miejscach jednocześnie

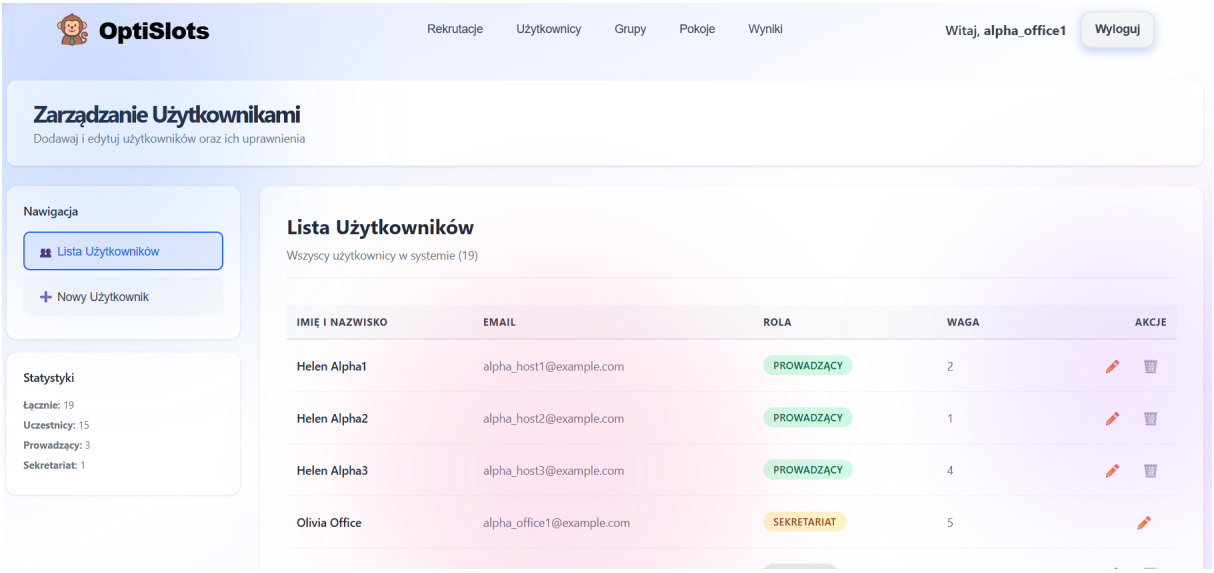

Figure 3: Widok frontendu administratora firmy - zarządzanie użytkownikami

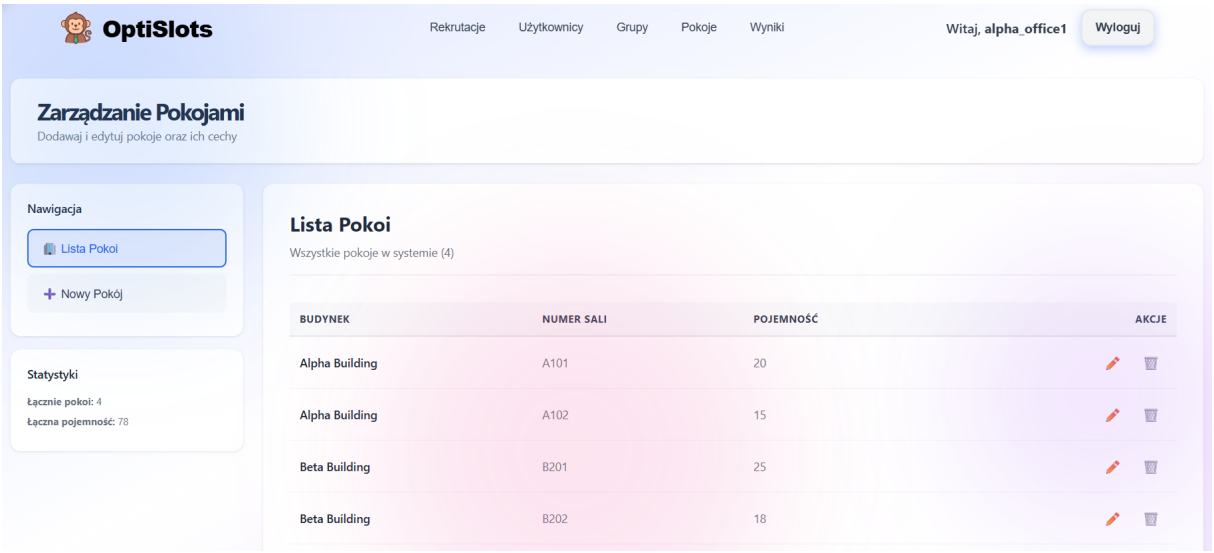

Figure 4: Widok frontendu administratora firmy - zarządzanie pokojami

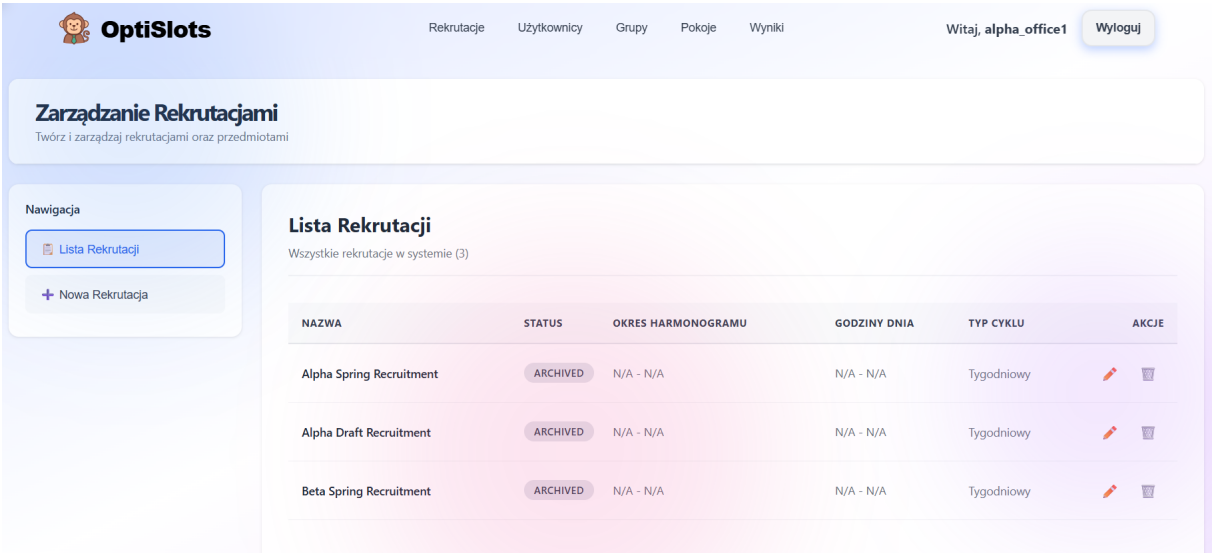

Figure 5: Widok frontendu administratora firmy - zarządzanie rekrutacjami

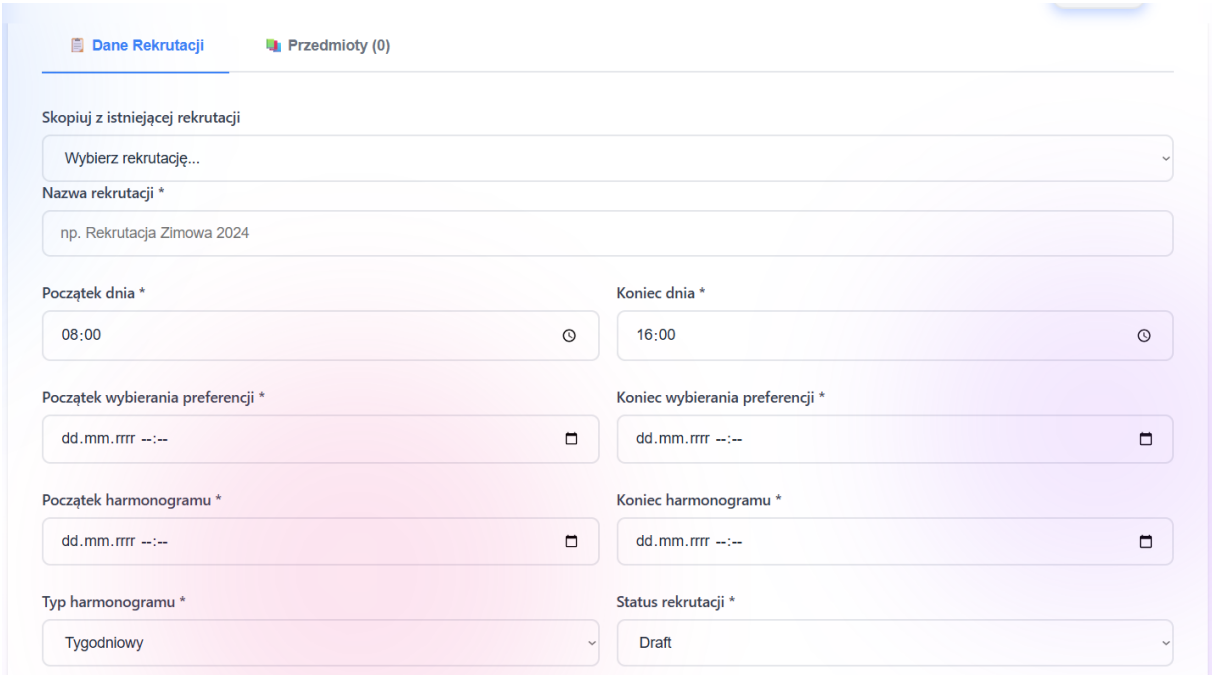

Figure 6: Widok frontendu administratora firmy - formularz tworzenia nowej rekrutacji

Użytkownicy administracyjni firmy ('office') mogą zarządzać użytkownikami [3], pokojami [4] oraz dodawać rekrutacje [6].System oferuje również dodatkowe udogodnienia - użytkownicy mogą należeć do wielu grup jednocześnie w celu ułatwienia przydzielania osób do rekrutacji lub przedmiotów. Pokoje mogą być opatrzone dowolnymi etykietami opisującymi ich właściwości, co umożliwia szybkie i jednoznaczne określenie, które z nich spełniają wymagania danego spotkania. Przykładowo, w sytuacji gdy spotkanie wymaga dostępu do projektora, można zdefiniować etykietę „posiada projektor", przypisać ją odpowiednim pomieszczeniom, a następnie oznaczyć ją jako wymaganą podczas konfiguracji procesu rekrutacji.

2.0.2 REST API - Django python [4]

ID	USERNAME 2	EMAIL ADDRESS	FIRST NAME	LAST NAME	ROLE 1
e71c79d1-3c75-485d-b88a-3a1545b929ff	alpha_part1	alpha_part1@example.com	Pat	Alpha1	Participant
166beed5-b5dd-4fe9-a4b8-a16afbbb2992	alpha_part10	alpha_part10@example.com	Pat	Alpha10	Participant
e7dc23ff-70f5-4cb7-86be-37851c42e851	alpha_part11	alpha_part11@example.com	Pat	Alpha11	Participant
c49aeecc-9380-419b-	alpha_part12	alpha_part12@example.com	Pat	Alpha12	Participant

Figure 7: Widok użytkowników w panelu administracyjnym Django

REST API, zbudowane w Django zarządza danymi systemu, odbierając dane od frontendu i formatując je do dalszego wykorzystania w procesach systemowych. Jest odpowiedzialny za nadawanie i weryfikację uprawnień dla logujących się użytkowników, udostępnianie zapytań API dla innych komponentów oraz wykonywanie procesów potrzebnych do płynniejszego działania systemu.

2.0.3 Optymalizator - C++ [7]

Optymalizator jest odpowiedzialny za weryfikację rozwiązywalności problemu i ułożenie optymalnego rozwiązania w podanym okresie czasowym. Po otrzymaniu danych od modułu pomocniczego optymal-

```
[DEBUG] Iteration: 3/10
[DEBUG] Iteration: 4/10
[DEBUG] Iteration: 5/10
[DEBUG] Iteration: 6/10
[DEBUG] Iteration: 7/10
[DEBUG] Iteration: 8/10
[DEBUG] Iteration: 9/10
[INFO] Iteration 1, best fitness: 0.689486
[INFO] Sending progress to Redis for job 5d5aece2-0135-4417-ad45-5636ddc03d06, iteration 1
[INFO] Successfully sent progress for job: 5d5aece2-0135-4417-ad45-5636ddc03d06 (notified 1 subscribers)
[INFO] Progress message: {"best_solution":{"by_group":[[84,88,0],[73,77,0],[151,155,0],[98,102,1],[172,176,0],[104,108,1],
[0,4,0],[4,8,0]],"by_student":[[4,6,5],[7,6,5,0],[2,7,1],[2,1,4],[2,7,6,0],[2,1,4,5],[],[2,1,5,3],[],[2,1,4,5],[2,7,3],[],
[],[],[1,6,5,0]],"days_in_cycle":7,"fitness":0.6894860930203774,"genotype":
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,84,0,73,0,151,0,98,1,172,0,104,1,0,0,4,0],"student_
detailed_fitnesses":[[[0.5714285714285714,1.0],[1.0,0.0],[1.0,0.0],[0.1428571428571429,1.0],[1.0,2.0],[1.0,5.0],[1.0,3.0],
[1.0,10.0],[1.0,0.0],[0.8541666666666666,14.0],[1.0,0.0],[0.0,27.0],[0.13043478260869565,23.0]],[[0.5714285714285714,1.0],
[0.8333333333333334,2.0],[1.0,0.0],[0.4285714285714286,1.0],[1.0,2.0],[1.0,1.0],[1.0,3.0],[1.0,10.0],
[0.7395833333333333,5.0],[0.8020833333333334,11.0],[1.0,0.0],[0.0,17.0],[0.6956521739130435,23.0]],
[[0.5714285714285714,3.0],[0.875,2.0],[1.0,0.0],[1.0,0.0],[1.0,2.0],[1.0,2.0],[1.0,0.0],[1.0,9.0],[0.6875,7.0],
[0.7604166666666666,12.0],[1.0,0.0],[0.0,28.0],[0.52,25.0]],[[0.5714285714285714,5.0],[0.875,1.0],[1.0,2.0],
[0.4285714285714286,3.0],[1.0,2.0],[1.0,3.0],[1.0,3.0],[1.0,7.0],[0.6354166666666667,1.0],[0.8333333333333334,8.0],
```


Figure 8: Widok działania optymalizatora

izator zaczyna turowo rozwiązywać problem, zwracając po każdej turze najlepszy plan do wglądu przez administrację firmy. Pomiędzy turami optymalizator otrzymuje aktualne preferencje użytkowników, zapewniając satysfakcje uczestników i prowadzących.

2.0.4 Moduły pomocnicze

redis-1	44361b9d9676	redis:7-alpi	6379:6379
progress-listener-1	cb17e1e94996	zpi-evolutic	
scheduler-1	01bdd5770dba	zpi-evolutic	

Figure 9: Moduły pomocnicze

W systemie występują cztery moduły pomocnicze:

- Baza danych - przechowuje wszystkie dane systemowe
- Scheduler - monitoruje warunki uruchomienia procesu optymalizacji harmonogramu i, gdy są spełnione, przekazuje do optymalizatora nową lub zaktualizowaną formę planu.
- Progress listener - po każdej turze optymalizatora przesyła najlepszą wersję harmonogramu do komponentu Django, zapewniając klientowi aktualne informacje o postępie tworzenia planu.
- Redis - pełni funkcję brokera wiadomości, kolejkując i przekazując komunikaty między optymalizatorem a komponentem Django.

3 PODSUMOWANIE

3.1 Wnioski

Zrealizowany projekt potwierdził, że zastosowanie optymalizacji ewolucyjnej w procesie tworzenia harmonogramów pozwala znacząco go usprawnić. Powstały system prawidłowo przetwarza i kolejkuje dane tworząc plany nie łamiące ograniczeń definiowanych przez administrację i spełniające preferencje wypełniane przez użytkowników. Generowane plany charakteryzują się wyższą jakością (wyrażoną stopniem realizacji preferencji) w stosunku do rozwiązań początkowych.

Dla użytkowników biznesowych projekt ma kluczowe znaczenie, ponieważ umożliwia efektywne zarządzanie zasobami przy mniejszym nakładzie pracy i większej przewidywalności rezultatów. Automatyzacja procesu planowania przekłada się na redukcję kosztów operacyjnych, szybsze przygotowywanie harmonogramów oraz większą przejrzystość całego procesu. Dzięki integracji preferencji użytkowników firma uzyskuje rozwiązania, które lepiej odpowiadają rzeczywistym potrzebom zespołów, co zwiększa satysfakcję zarówno pracowników, jak i klientów.uczestników.

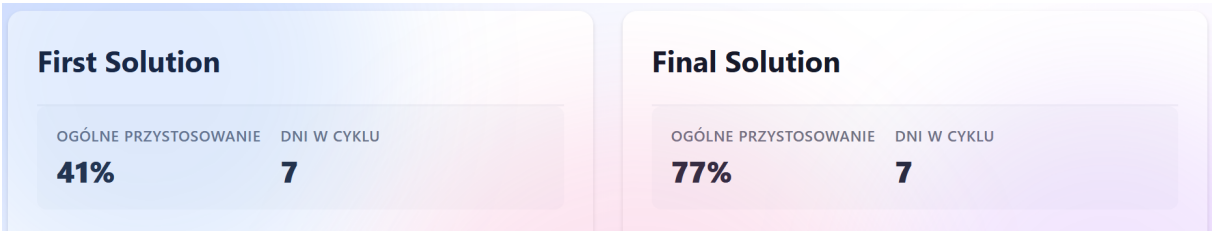

Figure 10: Porównanie wyników przed i po optymalizacji

3.2 Kolejne kroki

System spełnia swoje zadanie, ale może być dalej ulepszony, poprawiając wydajność i wygodę użytkowania. Jednym z elementów którego implementacja lezała w naszych planach to zapewnienie większej ilości preferencji i udogodnień np.:

- Sposób na wykluczenie godzin przez użytkownika - aktualnie w naszym systemie jest to możliwe, ale wymaga stworzenia pustego spotkania z nim przez administrację co jest dość nieintuicyjne.
- Zaawansowana heatmapa - aktualna heatmapa wyświetla aktualne zainteresowanie okresami czasu; w przyszłości mogłaby wyświetlać bardziej skomplikowane preferencje innych użytkowników. Dało by to użytkownikom więcej wglądu w w wybory innych członków, poprawiając ogólną jakość planu
- Importowanie danych ze źródeł zewnętrznych - system umożliwiałby importowanie danych o użytkownikach i pokojach z zewnętrznych źródeł, takich jak USOS lub pliki CSV. Przyspieszyłoby to proces ustalania struktury firmy.
- Persystencja stanu - aby zachować ciągłość obliczeń pomiędzy dwiema kolejnymi iteracjami obliczeń serwisu optymalizującego, kluczowe jest przechowywanie najlepszego rozwiązania bądź całej populacji rozwiązań w bazie danych, tak aby kolejny agent podejmujący się zadania na potencjalnie nowych preferencjach, a przez to innej hierarchii jakości rozwiązań mógł kontynuować od tego samego punktu, na którym skończył pierwszy agent.

REFERENCES

[1] Appknox. How jwt helps in securing your api. `https://www.appknox.com/blog/how-jwt-helps-in-securing-your-api`.

[2] Calendly. System calendly. `https://calendly.com`.

[3] ClickUp. System clickup. `https://clickup.com`.

[4] Django Software Foundation. Django documentation. `https://www.djangoproject.com`.

[5] Evan You et al. Vue.js documentation. `https://vuejs.org`.

[6] Google. System google calendar. `https://calendar.google.com`.

[7] International Organization for Standardization. C++ documentation. `https://devdocs.io/cpp/`.

[8] Leslie Lamport. *LaTeX: A Document Preparation System*. Addison-Wesley, 1994.

[9] PostgreSQL Community. Postgresql (pl) website. `https://www.postgresql.org.pl`.

[10] Postman, Inc. Postman documentation. `https://www.postman.com`.

[11] Redis, Inc. Redis documentation. `https://redis.io`.

[12] SQLite Consortium. Sqlite documentation. `https://sqlite.org`.

[13] University Study-Oriented System (USOS). System usos. `https://usos.edu.pl`.

[14] Vercel. Next.js documentation. `https://nextjs.org`.