01.06.2020

Usługi Sieciowe w Biznesie

Projekt

REST API w technologii Flask

Bartłomiej Nawój,

156314, L05

Spis treści

1. Wstępny opis projektu 3
Ogólne założenia projektu 3
Użyte technologie 3
2. Analiza techniczna 4
Diagram ERD 4
Diagramy aktywności dla funkcjonalności 5
3. Platformy do wdrażania aplikacji 7
DigitalOcean 7
Heroku 8
4. Testowanie za pomocą Postman 9
Endpointy 9
5. Podsumowanie projektu 11

1. Wstępny opis projektu

Ogólne założenia projektu

Celem projektu było stworzenie API, które symulowałoby system kontroli pracowników. Zaproponowane rozwiązanie imituje system ewidencji czasu pracy oparty o karty dostępu. Takie rozwiązanie stosowane jest dzisiaj w wielu firmach, gdzie istnieje potrzeba zwiększenia bezpieczeństwa (kontrola dostępu do budynku) połączonego z ewidencją czasu pracy.

Oprócz zaprojektowania i implementacji systemu, założeniem projektu było porównanie dostępnych opcji wystawienia API w świat oraz przetestowanie go za pomocą profesjonalnego narzędzia.

Użyte technologie

Python

Interpretowany język wysokiego poziomu o bardzo szerokim zastosowaniu. Łatwa do zrozumienia składnia oraz jedna z najbogatszych baz zewnętrznych bibliotek czyni ten język popularnym w wielu dziedzinach: od tworzenia aplikacji webowych po data science.

Flask

Mikro-framework aplikacji webowych napisany w języku Python. Mikro, ponieważ Flask nie zawiera w sobie wbudowanych rozwiązań autentykacji, warstwy obsługującej bazy danych itd. Zamiast tego bazuje na szerokiej bazie modułów, których wybór należy do programisty. Taka elastyczność jest ogromną zaletą. Flask jest łatwy w konfiguracji i prosty w obsłudze. Swoje zastosowanie znalazł w firmach takich jak LinkedIn, Uber czy Netflix.

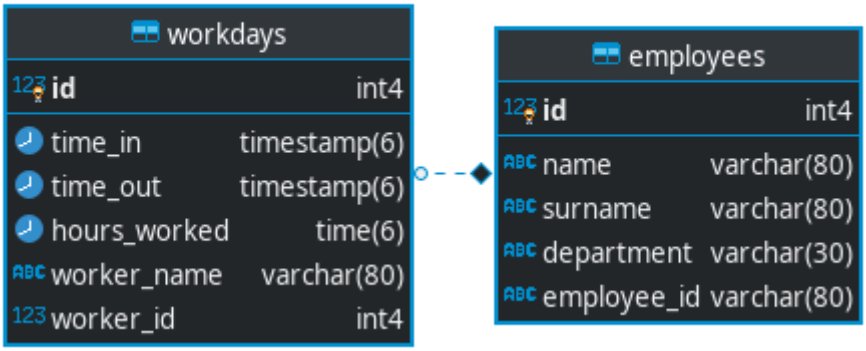
2. Analiza techniczna

Działanie systemu opiera się o zbudowane we Flasku REST api. Architektura REST umożliwia odseparowanie warstwy klienta od warstwy serwerowej. Zaletą takiego rozwiązania jest jego uniwersalność. Ze stworzonego API może korzystać wiele różnych "klientów" takich jak strony internetowe czy aplikacje na urządzenia mobilne.

Do przechowywania informacji o pracownikach i czasie pracy zastosowano bazę danych PostgreSQL. Pracę z bazą danych ułatwiono dzięki Flask-SQLAlchemy (odpowiednik SQLAlchemy przystosowany do pracy z frameworkiem Flask). Jest to moduł służący do mapowania obiektowo relacyjnego.

Diagram ERD

Diagram związków encji (ang. Entity-Relationship-Diagram) dla stworzonej bazy danych wygenerowany za pomocą programu DBeaver:



W tabeli **employees** przechowywane są następujące dane pracowników firmy:

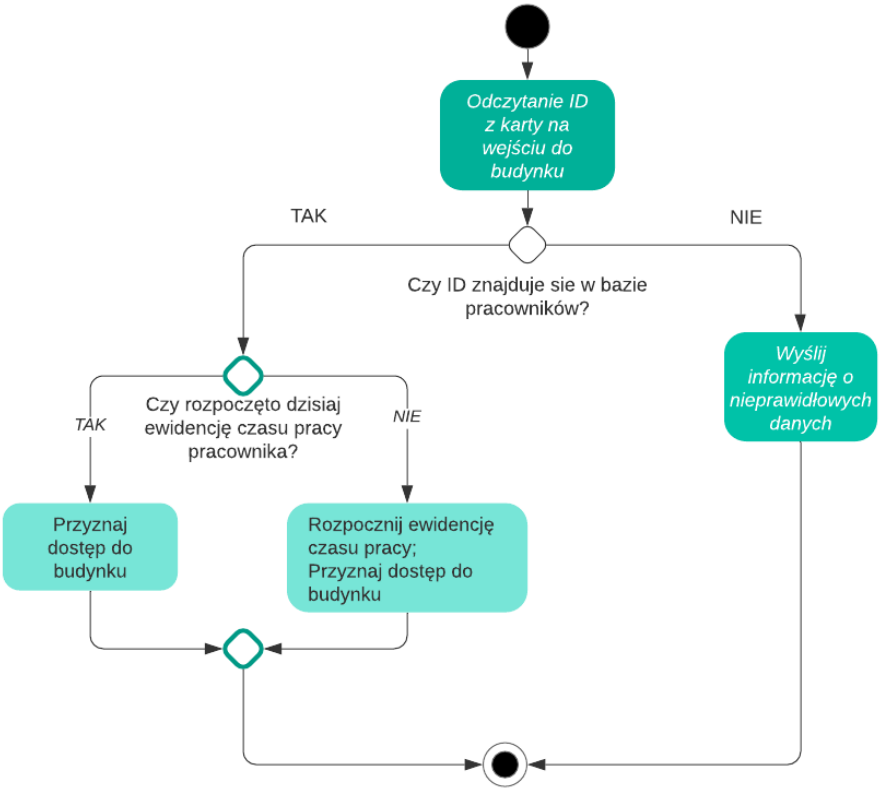
- **name** – imię pracownika
- **surname –** nazwisko pracownika
- **department –** dział zatrudnienia
- **employee_id –** unikatowy id (uuid) symulujący ID przypisany do karty identyfikacyjnej

Tabela **workdays** przechowuje dane potrzebne do prowadzenia ewidencji dnia pracy:

- **time_in –** moment wejścia pracownika do firmy
- **time_out –** moment wyjścia pracownika z firmy
- **hours_worked –** obliczony czas pracy danego dnia
- **worker_name –** imię pracownika
- **worker_id** – klucz obcy wskazujący na employees.id

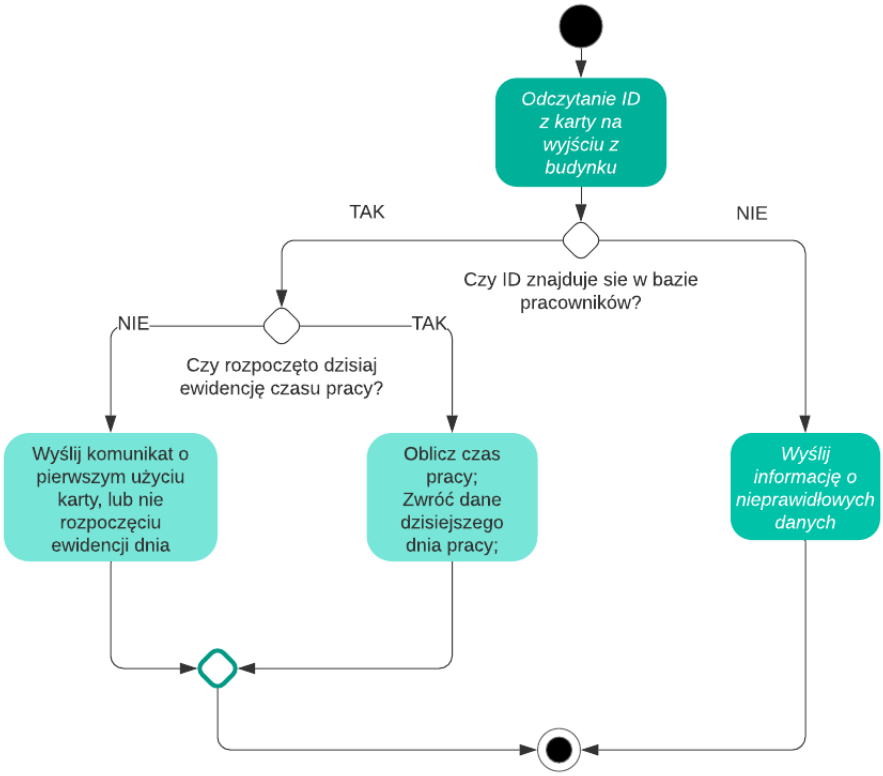
Diagramy aktywności dla funkcjonalności

Przyznawanie dostępu i rozpoczynanie ewidencji czasu pracy.



Pracownik ma możliwość wejścia do budynku, jeśli ID jego karty znajduje się w bazie danych w tabeli **employees**. Po pomyślnej weryfikacji serwer zwraca **access_token** oraz rozpoczyna ewidencję dnia pracy tworząc odpowiedni wpis w tabeli **workdays**. Ewidencja jest rozpoczynana tylko raz dziennie. W przypadku ponownego wejścia do budynku, po pomyślnej weryfikacji ID, zwracany jest tylko **access_token.**

Opuszczanie budynku i obliczanie przepracowanych godzin



Przy opuszczaniu miejsca pracy pracownik używa swojej karty do otworzenia drzwi. Po prawidłowej weryfikacji, jeśli pracownik zaczął dzisiaj ewidencję czasu pracy, serwer oblicza czas pracy i umieszcza go odpowiednio w tabeli **workdays**. Przypadki pierwszego użycia karty oraz użycia karty, która nie jest przypisana do żadnego z pracowników, zostały odpowiednio zabezpieczone.

Do celów testowych stworzono również funkcjonalność dodawania nowych pracowników do bazy. Podczas dodawania pracownika, każdy z nich otrzymuje 36 znakowy unikatowy ID (uuid) co symuluje przypisanie nowej karty identyfikacyjnej. W tym samym celu możliwe jest również pobranie informacji o pracowniku.

3. Platformy do wdrażania aplikacji

Istnieje wiele możliwości wdrażania (ang. deployment) aplikacji. Różnią się stopniem skomplikowania procesu wdrażania, udogodnieniami, możliwościami konfiguracji serwera (lub jej ograniczeniem), oraz oczywiście kosztami. Żadna z platform nie jest idealna dla każdego projektu. Wybór powinien być podejmowany biorąc pod uwagę aktualne potrzeby, ponieważ każda opcja ma swoje zalety. W tym rozdziale podjęto próbę porównania dwóch popularnych platform.

DigitalOcean

Jest to platforma chmurowa typu IaaS (Infrastructure as a Service) stworzona z myślą uproszczenia procesu wdrażania skierowana głównie do webdeveloperów. Umożliwia ona wypożyczanie serwerów o różnych konfiguracjach sprzętowych w zależności od opłat. Platforma zapewnia gotowe wzory maszyn z popularnym frameworkami, aplikacjami i gotowymi środowiskami typu LAMP. Do dyspozycji programistów, DigitalOcean udostępnia API do zarządzania maszynami oraz przejrzysty panel kontrolny.

W odróżnieniu od Heroku otrzymujemy dostęp do fizycznego komputera (tutaj nazwany Dropletem) który służy nam za serwer. Umożliwia to wszelkie operacje takie jak:

- wybór systemu dla naszego serwera,
- zmiany użytkowników,
- dodanie większej ilości programów działających razem z naszą aplikacją,
- zmiana parametrów bezpieczeństwa, itd..

Dzięki bogatym opcjom konfiguracji możliwe jest obniżenie kosztów utrzymania aplikacji, dopasowanie serwera do naszych indywidualnych potrzeb oraz łatwe skalowanie w przypadku ewentualnego rozwoju aplikacji. Wiąże się to jednak z większym poziomem skomplikowania i zaawansowania procesu wdrażania.

Koszty miesięczne oraz koszty za godzinę wynajmowania serwera na DigitalOcean w zależności od parametrów maszyny i dostępnego transferu:

Memory	vCPUs	Transfer	SSD Disk	$/HR	$/MO
1GB	1vCPU	1TB	25GB	$0.007	$5
2GB	1vCPU	2TB	50GB	$0.015	$10
3GB	1vCPU	3TB	60GB	$0.022	$15
2GB	2vCPUs	3TB	60GB	$0.022	$15
1GB	3vCPUs	3TB	60GB	$0.022	$15
4GB	2vCPUs	4TB	80GB	$0.030	$20
8GB	4vCPUs	5TB	160GB	$0.060	$40
16GB	6vCPUs	6TB	320GB	$0.119	$80
32GB	8vCPUs	7TB	640GB	$0.238	$160
48GB	12vCPUs	8TB	960GB	$0.357	$240
64GB	16vCPUs	9TB	1,280GB	$0.476	$320
96GB	20vCPUs	10TB	1,920GB	$0.714	$480
128GB	24vCPUs	11TB	2.5TB	$0.952	$640
192GB	32vCPUs	12TB	3.75TB	$1.429	$960

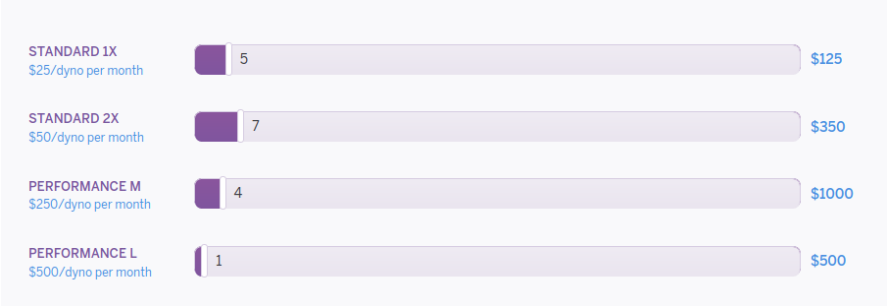
Heroku

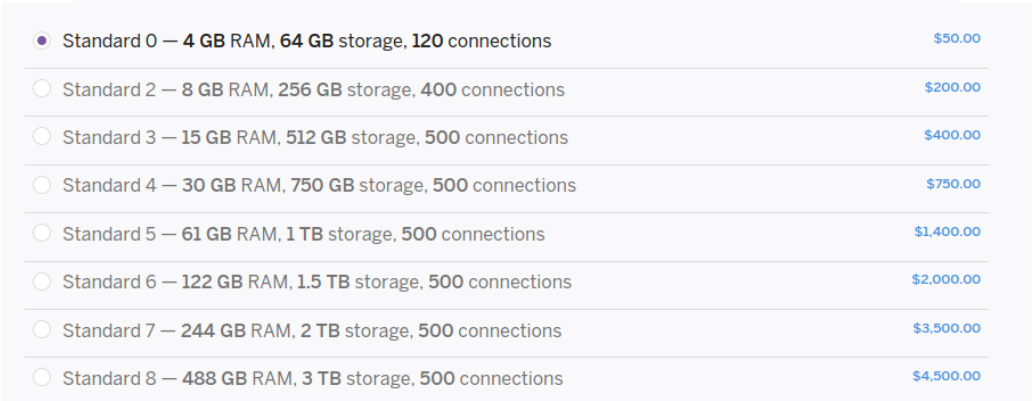
Platforma chmurowa jako usługa (ang. PaaS – Platform as a Service). Nie jest serwisem wypożyczającym serwery jak DigitalOcean. Tutaj nie trzeba się martwić infrastrukturą - koncentracja developera pozostaje na tworzeniu aplikacji, a sam procesu wdrażania jest prosty i natychmiastowy. “Serwery” - a raczej kontenery nazwane tutaj Dyno - uruchamiają naszą aplikację, lecz nie mamy dostępu do fizycznej maszyny jak w przypadku DigitalOcean.

Najważniejsze cechy Heroku:

- Integracja z repozytorium na Github, co pozwala na wdrożenie aplikacji automatycznie podczas git push (Heroku śledzi zmiany wybranej gałęzi repozytorium)
- Obsługa popularnych języków programowania i frameworków takich jak Java, Python, Go, Scala, Ruby, Node.js, PHP, Clojure
- Uruchamianie aplikacji z pomocą inteligentnych kontenerów (ang. Smart containers) “Dynos” - ciągle usprawnianych przez zespół Heroku
- Izolacja - każdy dyno jest całkowicie odizolowany od siebie
- Wiele dodatków (bazy danych, aplikacje związane np. z wiadomościami mail, czy przetwarzaniem obrazów)
- Skalowalność - by skorzystać z większej ilości Dynos, lub z szybszych wariantów należy jedynie przesunąć suwak w panelu administracyjnym

Łatwość wdrażania aplikacji przy pomocy Heroku niesie niestety ze sobą koszty (przy dużych projektach). Przykład, w jaki sposób rosną koszty z każdym dodanym Dyno, oraz zwiększaniu pojemności bazy:





Pomimo pierwszego wrażenia, iż serwis jest zaporowo drogi to znajduje on swoje zastosowania w biznesie. Kluczem jest dobór platformy pod nasze potrzeby – do mniejszych projektów Heroku nadaje się idealnie. Dlatego też do wdrożenia stworzonego RESTapi do kontroli pracowników użyto darmowej wersji tego rozwiązania. Wersja darmowa jest w pełni wystarczająca (10 tys. rekordów w bazie danych).

4. Testowanie za pomocą Postman

Postman to profesjonalne narzędzie do testowania API posiadające wiele przydatnych funkcji. API testujemy preparując i wysyłając żądania (ang. Request) na jego adres – w tym przypadku https://employee-tracker-api.herokuapp.com/

Możliwe jest zapisywanie żądań, tworzenie skryptów zapisujących zmienne do globalnego środowiska, które dodatkowo możemy udostępnić współtwórcom projektu w celu lepszej kooperacji.

Endpointy

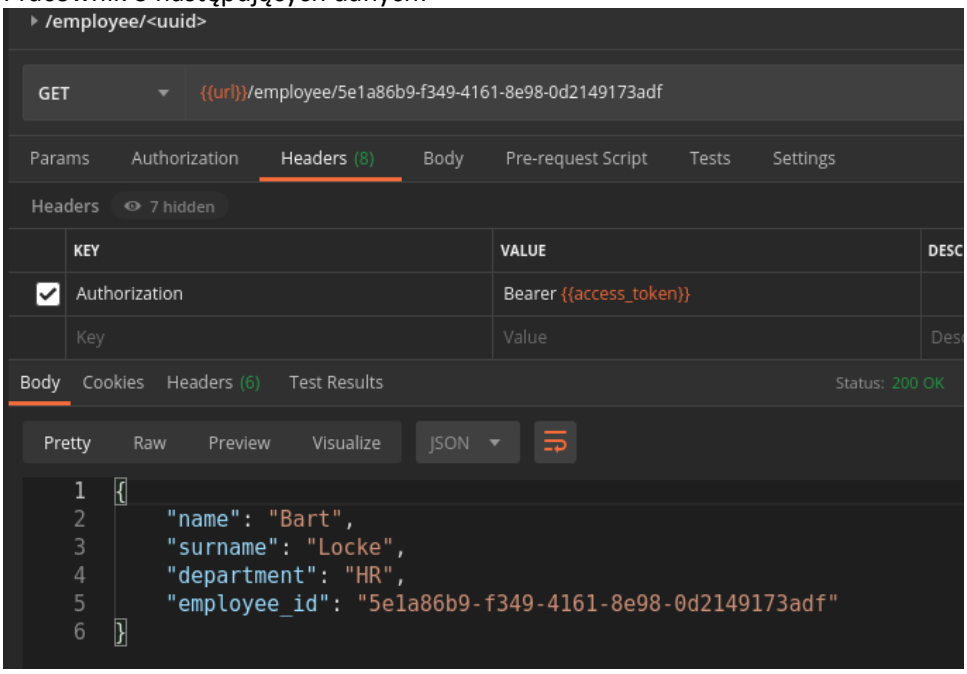
Aby zwiększyć czytelność późniejsze wykorzystywanie adresu API będzie zastąpione {{url}}.

1. **Przyznawanie dostępu i rozpoczynanie ewidencji czasu pracy.**
   W celu zapewnienia autentykacji API "employee-tracker" korzysta ze standardu JWT (JSON web tokens). Aby otrzymać token należy wykonać POST request na endpoint {{url}}/login.
   Ciało (ang. Body) żądania powinno być skonstruowane następująco:

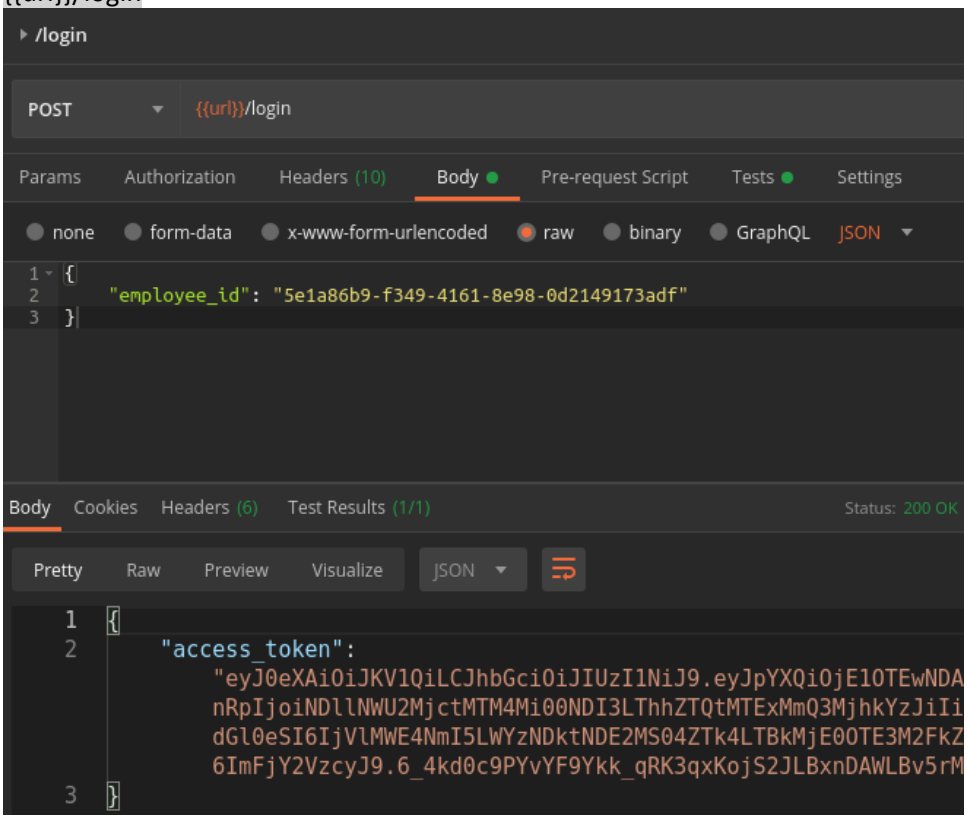
```
{
    "employee_id": "unikatowy-id-karty-pracownika"
}
```

   Przykład wykorzystania:

   - Pracownik o następujących danych:

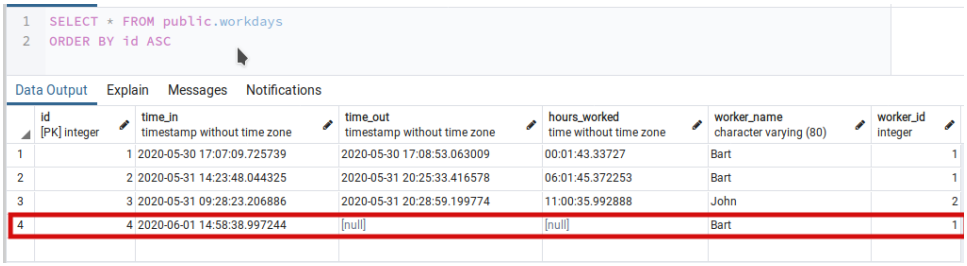


- przykłada kartę identyfikacyjną w celu wejścia do firmy – wysyłany jest POST request {{url}}/login



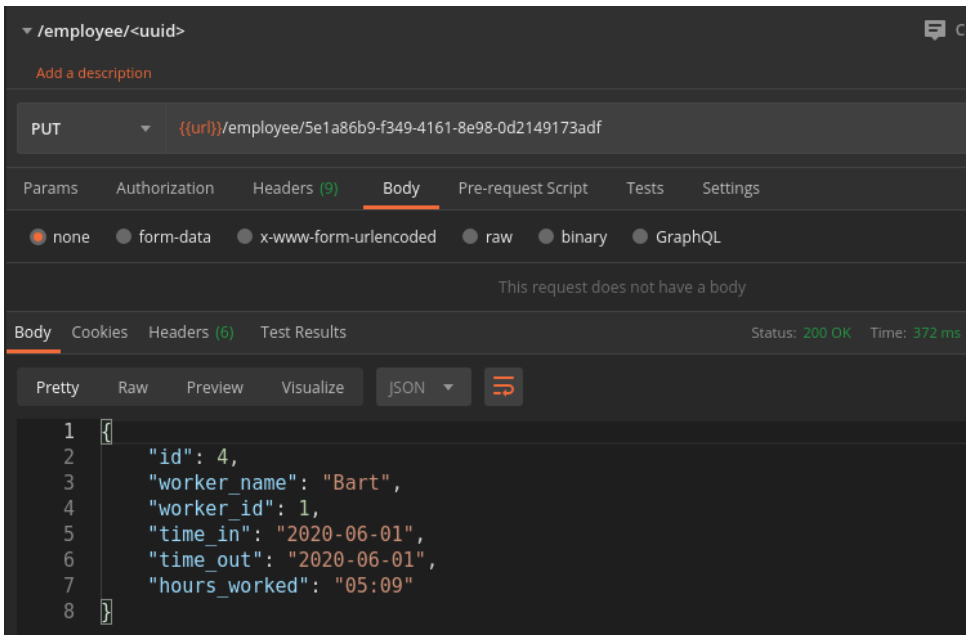
Jeśli dane były poprawne to w odpowiedzi zwracany jest access_token. Jeśli to było pierwsze wejście do firmy danego dnia, w bazie rozpoczynana jest ewidencja czasu pracy.

Zrzut ekranu z narzędzia administracyjnego bazy danych pgAdmin4 potwierdza poprawne działanie API. Utworzony wpis w bazie zaznaczono kolorem czerwonym:

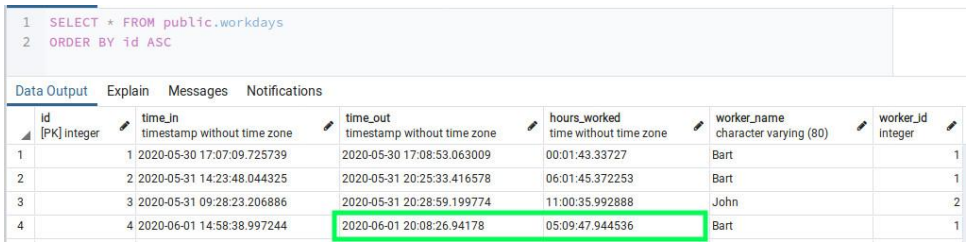


	id [PK] integer	time_in timestamp without time zone	time_out timestamp without time zone	hours_worked time without time zone	worker_name character varying (80)	worker_id integer
1	1	2020-05-30 17:07:09.725739	2020-05-30 17:08:53.063009	00:01:43.33727	Bart	1
2	2	2020-05-31 14:23:48.044325	2020-05-31 20:25:33.416578	06:01:45.372253	Bart	1
3	3	2020-05-31 09:28:23.206886	2020-05-31 20:28:59.199774	11:00:35.992888	John	2
4	4	2020-06-01 14:58:38.997244	[null]	[null]	Bart	1

2. **Opuszczanie budynku i obliczanie przepracowanych godzin**
   Przy wychodzeniu z budynku pracownik ponownie przykłada identyfikator do czytnika. Wysyłany jest PUT request na endpoint {{url}}/employee/<uuid>. Serwer zapisuje czas wyjścia, oblicza czas w pracy i umieszcza go w kolumnie **hours_worked.**
   Przykład działania na tym samym pracowniku:



Zaktualizowane pola w bazie danych zaznaczono zielonym kolorem:



	id [PK] integer	time_in timestamp without time zone	time_out timestamp without time zone	hours_worked time without time zone	worker_name character varying (80)	worker_id integer
1	1	2020-05-30 17:07:09.725739	2020-05-30 17:08:53.063009	00:01:43.33727	Bart	1
2	2	2020-05-31 14:23:48.044325	2020-05-31 20:25:33.416578	06:01:45.372253	Bart	1
3	3	2020-05-31 09:28:23.206886	2020-05-31 20:28:59.199774	11:00:35.992888	John	2
4	4	2020-06-01 14:58:38.997244	2020-06-01 20:08:26.94178	05:09:47.944536	Bart	1

5. Podsumowanie projektu

Projekt spełnia wszystkie początkowe założenia. Podczas jego tworzenia znaleziono wiele miejsc do usprawnień, dobrym pomysłem rozwoju byłoby stworzenie aplikacji klienckiej do przeglądania danych z systemu.
Projekt udowadnia możliwość zastosowania RESTapi w biznesowych rozwiązaniach. Dodatkowo tworzenie ich w języku Python za pomocą framework'u Flask czyni to zadanie stosunkowo nieskomplikowanym i szybkim w implementacji. Projekty tworzone w tych technologiach można dalej rozwijać i skalować.

Repozytorium projektu znajduje się w serwisie Github pod adresem:

https://github.com/Balowen/employee-tracker