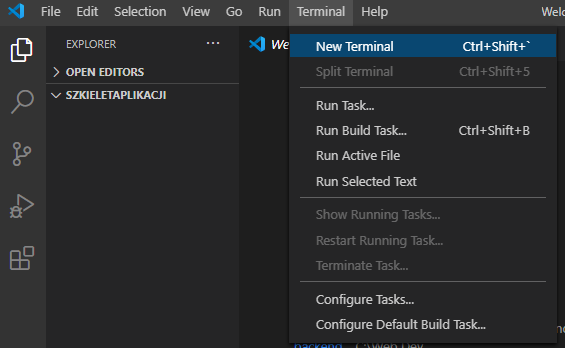
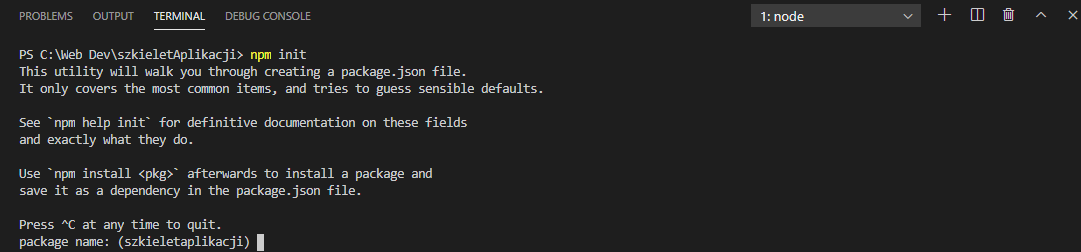
NodeJs + Express – szkielet aplikacji

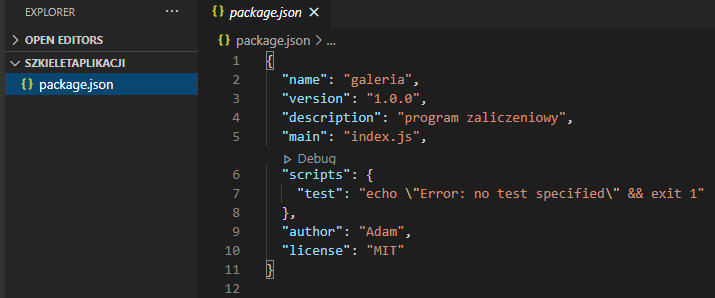
Przed rozpoczęciem pracy warto uruchomić Visual Studio Code i uruchomić terminal.



Aby zacząć projekt należy skorzystać z polecenia npm init

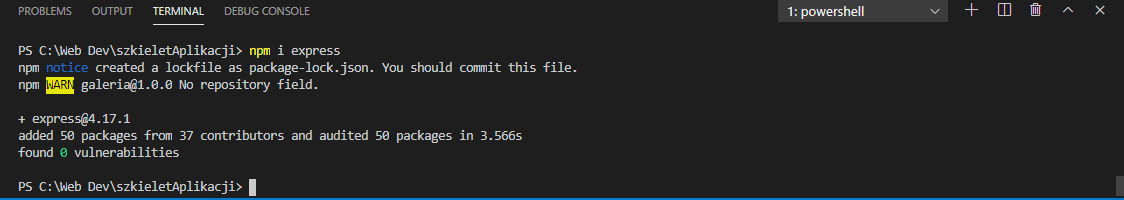


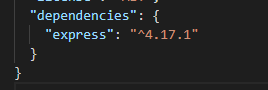
Po uruchomieniu polecenia należy ustalić nazwę aplikacji, opis, licencję, resztę poleceń można pominąć. Po wykonaniu powinien utworzy się plik package.json



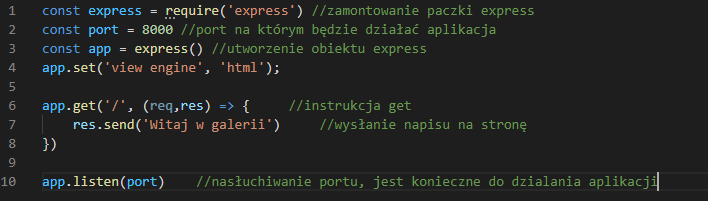
Plik ten nie musi być generowany za pomocą polecenia npm init, można równie dobrze stworzyć go samemu.

Kolejnym krokiem jest zainstalowanie pakietu Express. Do instalacji standardowo używamy npm.



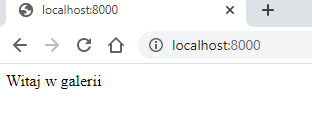
W tym przypadku polecenie zostało uruchomione poleceniem skróconym, można stosować „i” zamiast „install”. W tym momencie zostały utworzony plik package-lock.json dla nas nie istotny, oraz folder node\_modules zawierający potrzebne moduły. W pliku package.json został dodany parametr dependencies.  


Teraz kolejnym krokiem będzie utworzenie pliku index.js. Oczywiście można nazwać go inaczej jednak trzeba wówczas zmienić nazwę w pliku package.json.

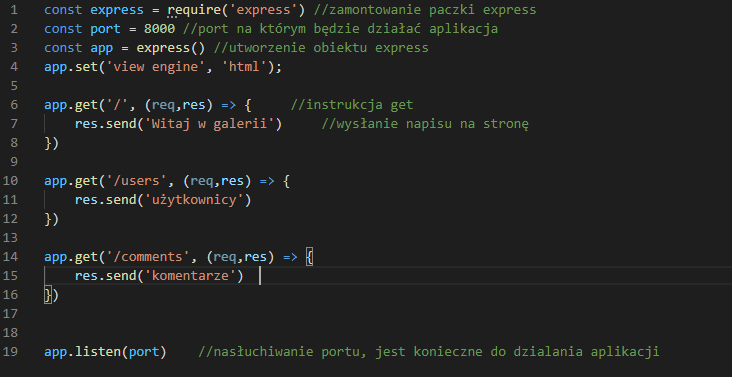


W powyższym kodzie app.get jest typową funkcją która przyjmuje req i res, nazwy skrócone od request i response. W tym przykładzie widać również nową formę zapisu funkcji przy użyciu strzałki. Linijka 6 oznacza że dla aplikacji Express dla metody Get, dla ścieżki głównej „/” czyli „localhost:8000/” i argumentami req i res wyślij napis „Witaj w galerii”.   
Taka instrukcja powinna wyświetlić napis „Witaj w galerii” po uruchomieniu poleceniem node index.js.

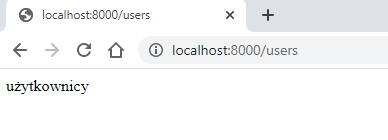




W tym miejscu mamy obsługę części CRUD a dokładnie Get, nie jest to jednak domyślne miejsce tego, ale przetestujmy działanie innych ścieżek.

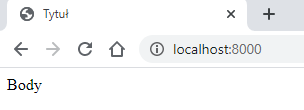


Aby te ścieżki zadziałały należy przerwać działanie serwera (Ctrl+c) i uruchomić na nowo komendą node index.js.

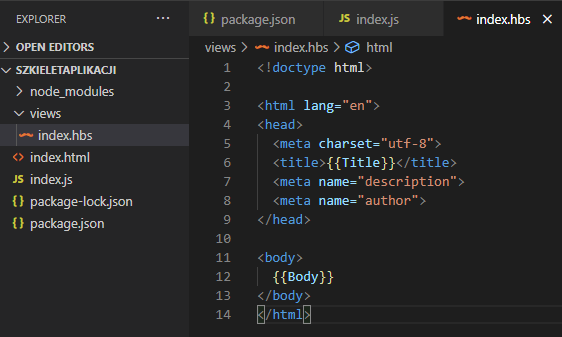


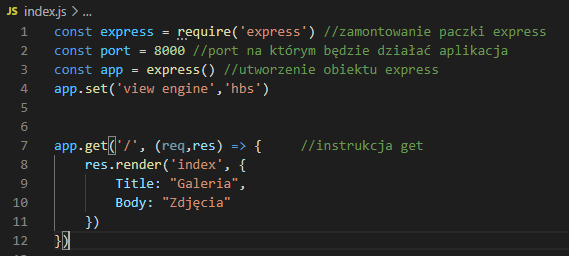
Generowanie napisu nie jest naszym celem, powinniśmy mieć możliwość wczytania pliku html. Dlatego res.send z linijki 7 możemy zmienić na sendFile, co pozwoli nam wczytać plik, który należy uprzednio stworzyć.

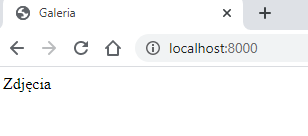




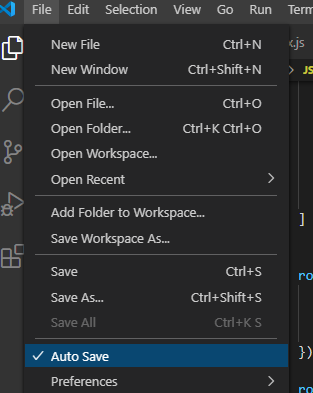
W tym momencie potrzebujemy przekazać coś do widoku, dlatego możemy skorzystać z paczki hbs, oczywiście jest do wyboru dużo innych paczek generowania widoku, ale w tym momencie możemy skorzystać z hbs (<https://www.npmjs.com/package/hbs>). Aby zainstalować paczkę wystarczy uruchomić instrukcję npm install hbs. Aby paczka zadziałał prawidłowo należy postąpić według instrukcji, a następnie utworzyć plik w folderze views.



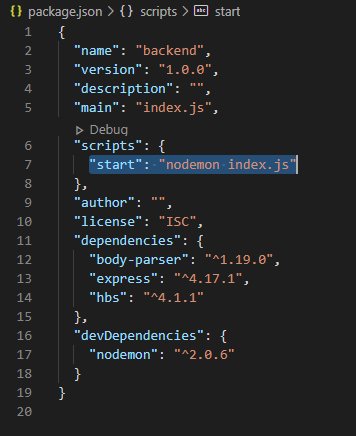




Możemy nieco pozmieniać aplikację i dodać więcej udogodnień. Program Visual Studio Code umożliwia automatyczny zapis. Dzięki temu nie trzeba zapisywać po każdej zmianie kodu, natomiast nasz serwer dalej trzeba odświeżać aby zaktualizować dane.



Aby nie trzeba było od nowa uruchamiać serwera po każdej zmianie, możemy wykorzystać moduł nodemon instalowany poleceniem: npm install --save-dev nodemon  
Aby uruchomić serwer należy dodać skrypt do pliku packane.json

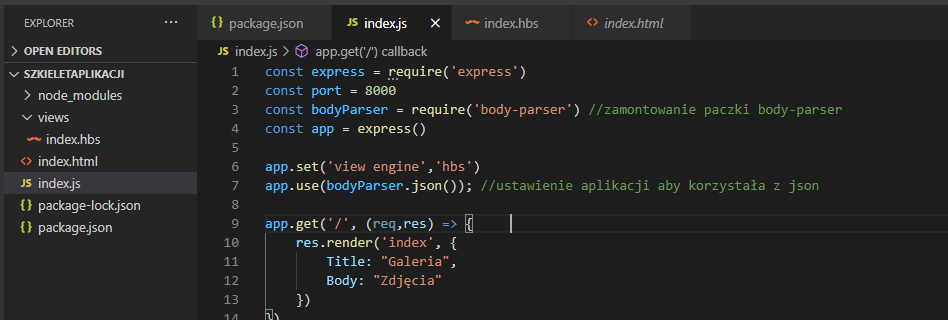


Kolejnym krokiem przydatnym do testowania komunikacji klient-serwer będzie dodanie modułu body-parser zgodnie z instrukcją: https://www.npmjs.com/package/body-parser. Po wykonaniu instalacji, package.json również uległ zmianie i do depenencies został dodany body-parser.

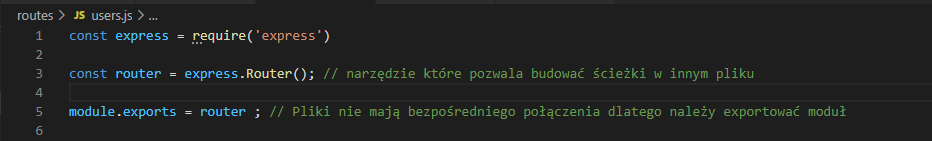
Po instalacji tych modułów, aby uruchomić aplikację wystarczy skorzystać z polecenia npm start, które uruchomi index.js za pomocą nodemon:



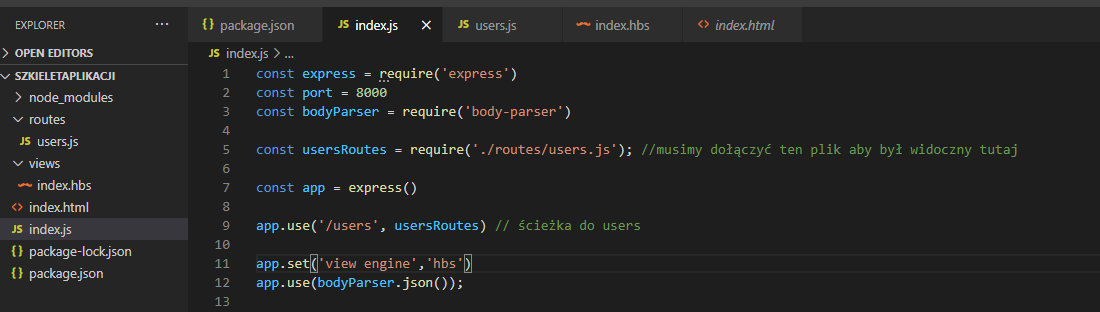
Aby skorzystać z modułu body-parser należy dodać moduł oraz ustawić używanie modułu dla plików json (dwie linijki z komentarzami)



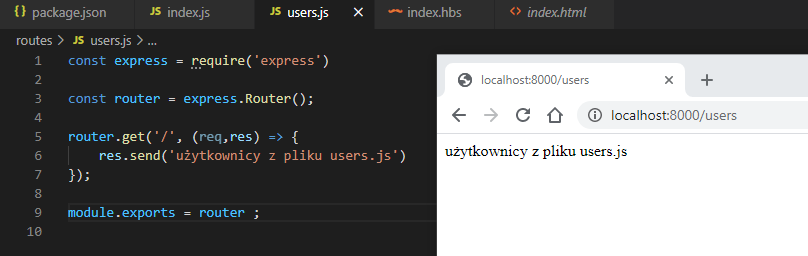
Teraz możemy stworzyć kolejne metody np. Post, ale w całej aplikacji będzie ich dużo dlatego warto posprzątać. Stwórzmy folder „routes” a wewnątrz plik users.js

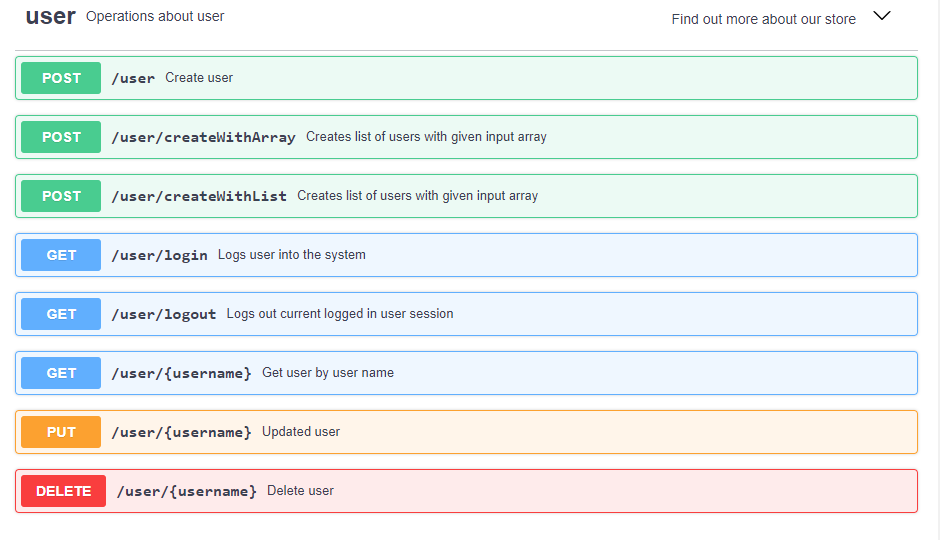


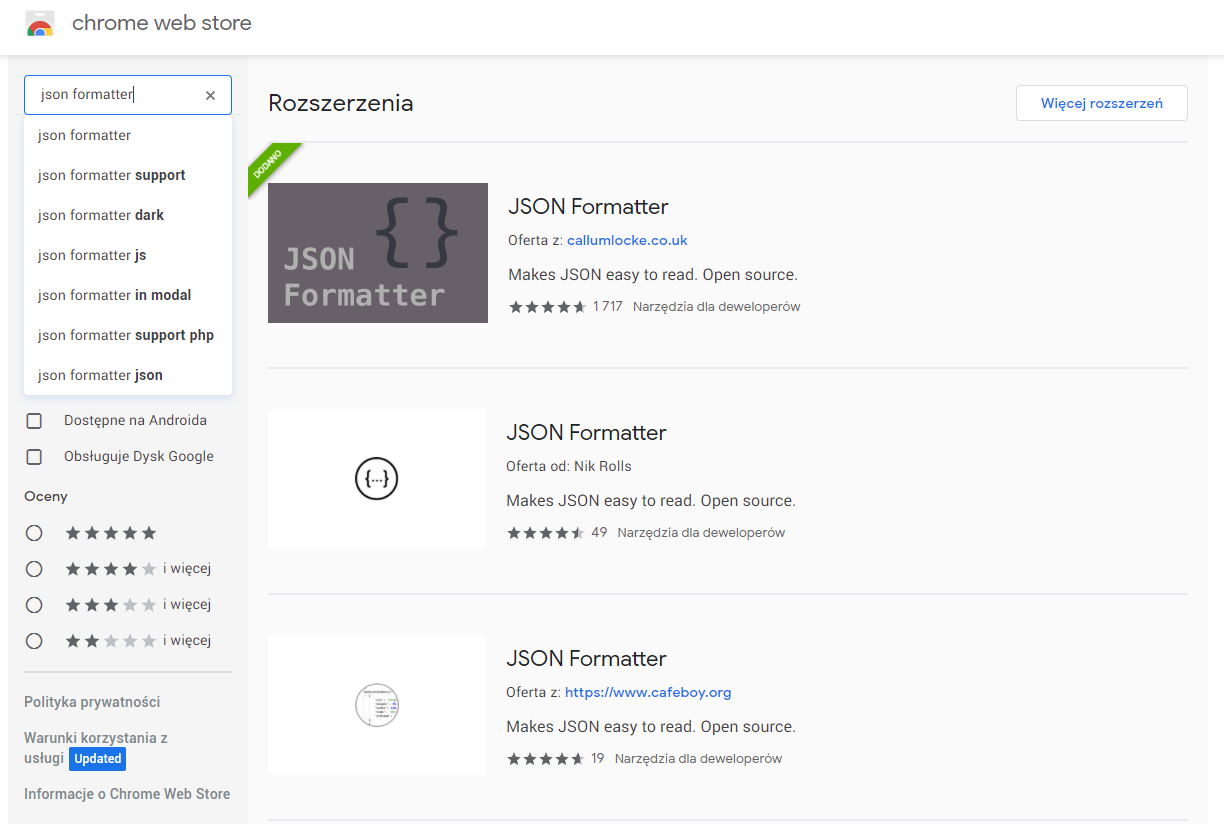
Dzięki temu będzie można uporządkować poszczególne ścieżki tak aby po „/users” można było bazować wewnątrz pliku users.js już bez używania nazwy „users”. Musimy zatem dodać pewnie instrukcje do pliku index.js aby ta ścieżka była widoczna.

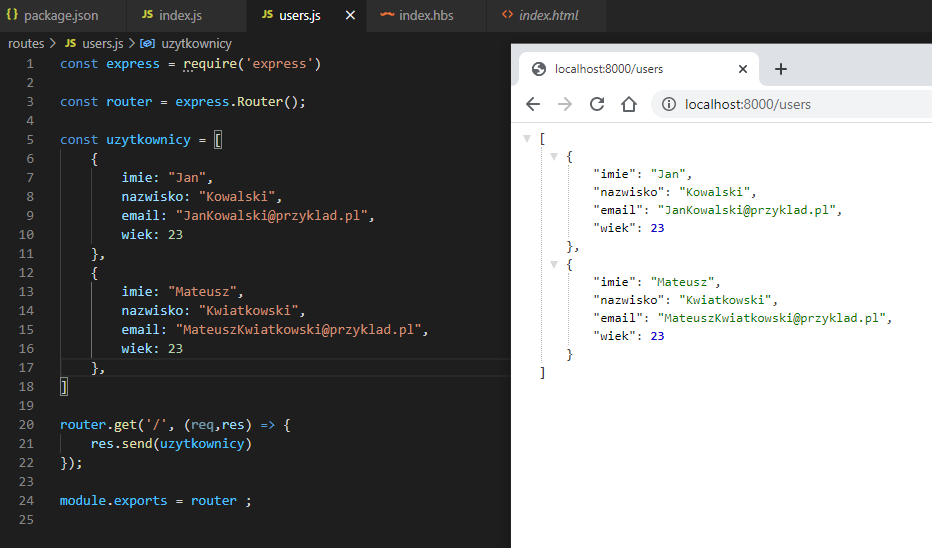


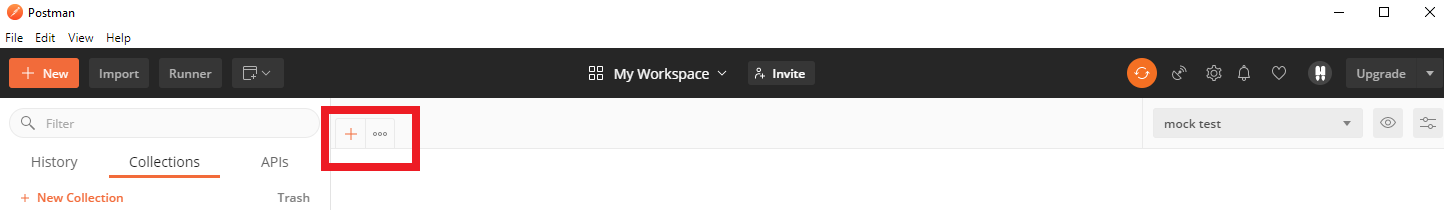
Aby przetestować ścieżkę wystarczy przesłać cokolwiek, np. napis za pomocą routera i funkcji get:



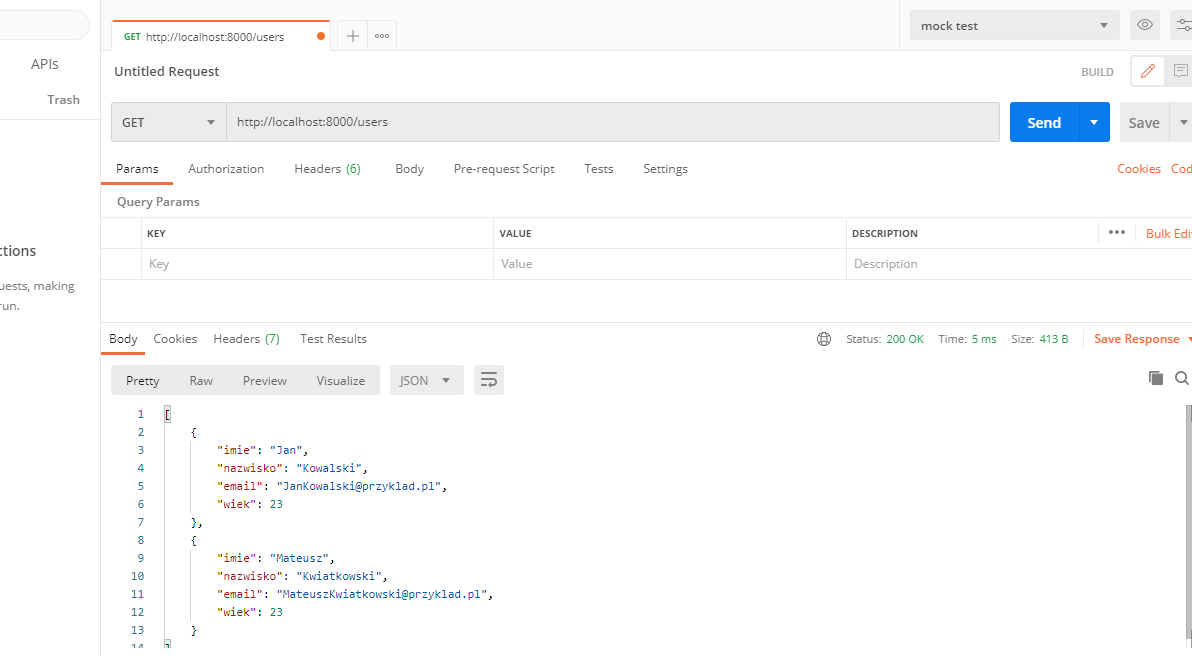
Możemy również stworzyć tymczasowe dane do testów, ponieważ nie posiadamy jeszcze bazy danych, ale na podstawie naszego API wiemy jakie metody powinniśmy utworzyć. Przykład wzięty z PetStore:  
Zatem korzystając z API utworzonego na poprzednich zajęciach możemy wstępnie zaimplementować metody, pamiętając że ścieżki już teraz zaczynają się od „/user” dlatego nie należy dopisywać tego wewnątrz pliku router/users.js. Do testu przygotujmy podstawową funkcję Get i Post która będzie przyjmowała i wyświetlała obiekty z tablicy sformatowanej podobnie do Json. Aby czytelnie widzieć dane wejściowe i wyjściowe można skorzystać z dodatków do przeglądarki. W przypadku chrome należy wybrać jeden z wielu dostępnych formaterów.



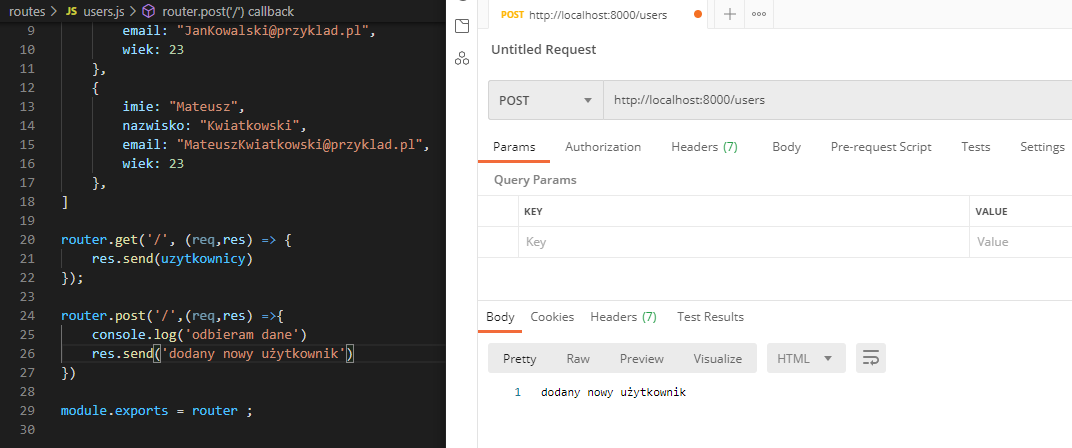
Na początku stwórzmy tablicę „użytkownicy” z polami imie, nazwisko, email, wiek, a następnie wyślij jako odpowiedź, czyli korzystając z res.  


Jak widać router zadziałał prawidłowo wysyłając tablicę zinterpretowaną jako Json w przeglądarce. Kolejnym krokiem będzie odebranie danych, oczywiście tutaj możemy mówić tylko o tymczasowym dodaniu nowego użytkownika, ponieważ nie mamy bazy danych. Zatem stwórzmy metodę Post, która jedynie poinformuje że coś otrzymała. Żeby zasymulować wysyłanie, należy pobrać i zainstalować aplikację „Postman”.  
Po uruchomieniu aplikacji pokaże się interfejs w którym należy dodać nowe okno:  


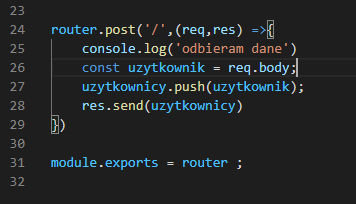
Po wybraniu powinien pojawić się pasek z wyborem metody i URL. Przetestujmy funkcję Get. Wystarczy wkleić adres naszego serwera, zaznaczyć metodę Get i wysłać, a poniżej otrzymujemy odpowiedź.

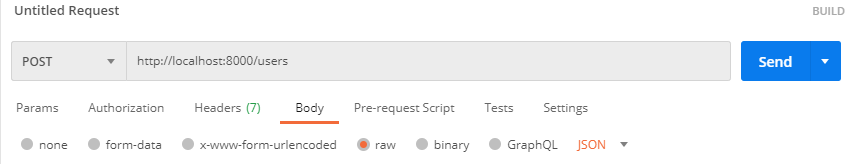


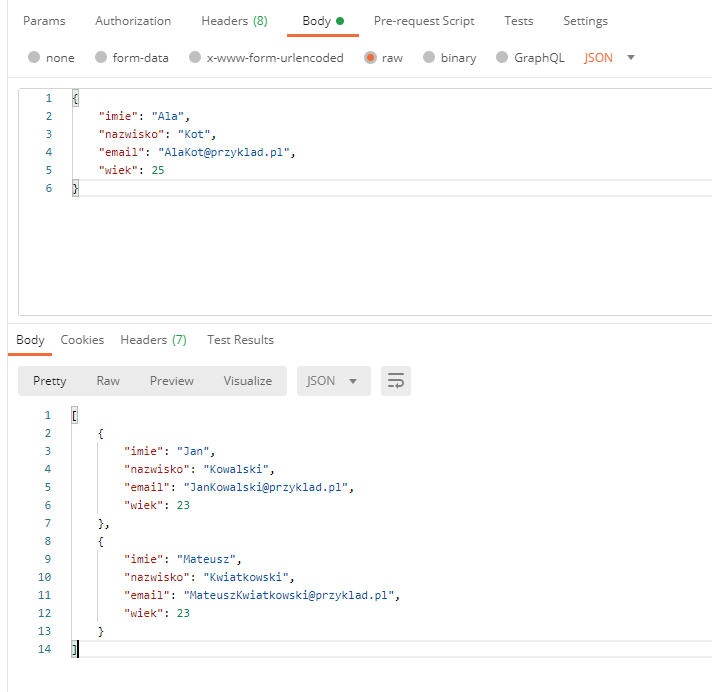
Status serwera 200, oraz wypisani użytkownicy sugerują że polecenie zadziałało poprawnie, dlatego przygotujmy funkcję Post.



Teraz spróbujmy faktycznie dodać użytkownika do naszej tablicy i wyświetlić.

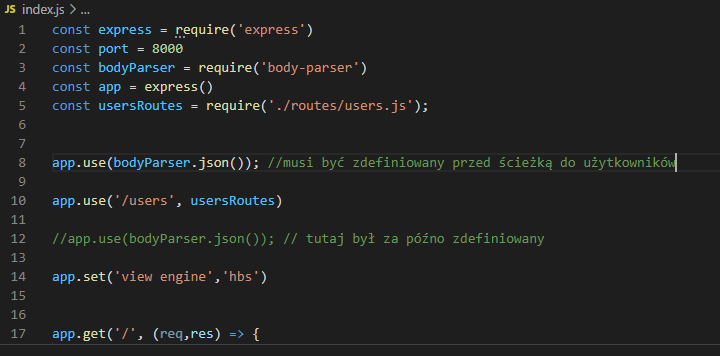


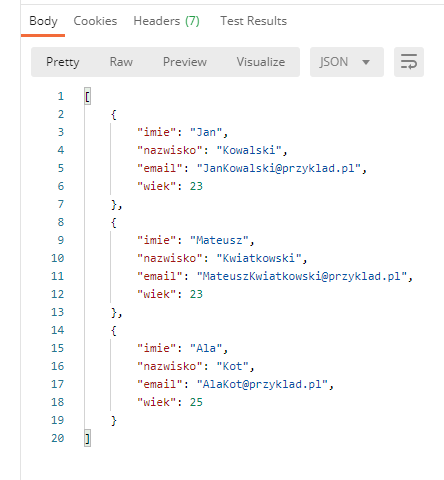
Aby skorzystać z Postmana musimy ustawić funkcję na POST, oznaczyć że dane będą wysyłane z Body, a następnie po prawej „text” zmienić na „JSON”  
  
Teraz wprowadźmy użytkownika i wyślijmy:



Po wykonaniu operacji został dodany nowy element:



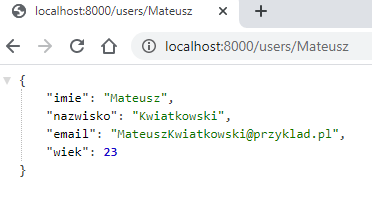
Jak widać nie jest on poprawny, dlatego jest błąd w kodzie, a dokładniej w body-parser.  


Po poprawieniu tego błędu widzimy że element został dodany:  
  


W kolejnym kroku możemy potrzebować odnieść się do konkretnego użytkownika czyli Get:id. Jako że w przykładzie nie ma ID możemy odnosić się np. do imienia, oczywiście nie jest to dobre rozwiązanie bo imion jest wiele, ale jako przykład przedstawię poniżej:



Req.params przyjmuje parametry które możemy później stosować w kodzie, polecam tutaj wypisać jak wygląda to w konsoli. W tym przypadku znajdujemy użytkownika w tablicy po id, którym jest imie. Aby odnieść się w przeglądarce wystarczy po ukośniku dopisać id.



**Zadanie.**

Zaimplementuj CRUD zgodnie z utworzonym własnym API dla diagramu User, a następnie przetestuj w Postmanie z danymi utworzonymi na sztywno. Dodatkowo stwórz pole id dla którego generowany będzie niepowtarzalny ciąg znaków. Wykorzystaj do tego dostępne moduły npm.