Odpowiedzi

TOOLS:

1) Myślę, że bardzo dobry. Z łatwością jestem w stanie nawigować po UI gitlaba. Mam pojęcie na temat CI/CD, potrafię skonfigurować workery na serwerach zdalnych (VPS) by nie musieć korzystać ze sparowanych workerów udostępnionych przez innych użytkowników tej platformy.
2) -
3) Tak, korzystałem ze Slite podczas pracy w GG International, poza tym przypadkiem nie miałem takiej potrzeby gdyż pracowałem solo.
4) Jak wyżej. Potrafię się po nim nawigować.
5) Korzystałem z OpenVPN z autoryzacją za pomocą certyfikatów w GG Int, ale również przy blokowaniu dostępu do innych projektów nad którymi pracowałem.
6) W głównej mierze korzystam z VSCode jako, że większość czasu pracowałem z Node/React, mam również znajomość Visual Studio IDE, Arduino IDE, PHPStorm oraz Eclipse. Nie tworzyłem nigdy snippetów ani templatek gdyż korzystałem ze snippetów udostępnionych przez innych użytkowników VSCode. Jednak gdyby brakowało mi jakichś skrótów bądź przyspieszyłoby to w jakimś stopniu moją pracę to na pewno bym się przymierzył do stworzenia własnych templatek i snippetów.
7) Korzystam z ESLint jako mojego głównego narzędzia do formatowania kodu. Trzymałem się raczej zasad i konfiguracji tego lintera udostępnionych przez Airbnb
8) W różnych sytuacjach podszedłbym do debugowania w inny sposób w zależności czy działałbym na frontendzie czy backendzie.
    a) W przypadku Reacta korzystam rozszerzenia chrome "React Developer Tools", który umożliwia w miarę łatwy debugging aplikacji wraz z podglądem wszystkich "state" oraz monitorowanie Javascriptowego event loopu oraz lifecycle otworzonych komponentów.
    b) W PHP wykorzystałbym var_dump($variable), print_r($variable), debug_zval_dump($variable) do wyświetlenia wartości danej zmiennej aby sprawdzić czy jest zgodna ze specyfikacją. (Czy jest taka jak być powinna w danym momencie), gdybym nie wiedział gdzie błąd występuje to na pewno skorzystałbym z rozszerzenia PHP Debug który wykorzystuje Xdebug i nawigował przez kod używając debuggera.
9) Do zarządzania bazami danych korzystam między innymi.
    a) MySQL Workbench (MySQL)
    b) MongoDB Compass (MongoDB)
    c) pgAdmin (PostgreSQL)
10) W przypadku MySQL wykorzystałbym narzędzie mysqldump do eksportu bazy danych do pliku .sql. Przy importowaniu zalogowałbym się do mysql na danej maszynie, stworzyłbym nową bazę danych a następnie importował przez zalogowanie się do mysql z dopisaniem do komendy `nazwa_bazy < baza.sql` a następnie sprawdziłbym czy dane zostały pomyślnie zaimportowane.
11) -
12) Używam rozszerzenia VSCode GitLens oraz wbudowanego 'Source Control' w VSCode do monitorowania commitów i historii zmian.

Deployment:

1. Composera znam, nie mam problemów z dodawaniem i usuwaniem bibliotek. Composer podczas uruchomienia composer install tworzy dodatkowy plik (composer.lock), który zapisuje i zabezpiecza wersje każdej biblioteki wykorzystywanej w projekcie. Dlatego później za każdym razem gdy nowa osoba dołącza do projektu to używając composer install instaluje te samą wersje wszystkich bibliotek jakie danych projekt wykorzystuje. Composer update ignoruje plik composer.lock i instaluje najnowsze wersje bibliotek zawartych w composer.json co może spowodować problemy z kompatybilnością kodu w przypadku jak któraś z zainstalowanych bibliotek zmieniła między wersjami swoje działanie lub składnie.
2. Tak, korzystam z npm, potrafię dodać, usunąć bibliotekę, zaktualizować bądź skompilować projekt korzystając z npm oraz yarn.
3.
    a. Skorzystałbym z 'merge', jak dla mnie jest to bezpieczniejsza opcja. W przypadku konfliktów jest możliwość ich rozwiązania lub całkowitego porzucenia merge. Jest też bardziej informacyjna gdyż wiemy skąd pochodzą zmiany.
    b. -
    c. Przy wykorzystaniu 'git revert', który cofnie dany branch do konkretnego commita.
    d. Merge conflict powstaje gdy dwa branche edytują tą samą linie kodu w tym samym pliku. Do tej pory rozwiązywałem takie problemy manualnie przez wybranie zmiany z jednego branchu lub drugiego lub przepisanie zmian w sposób aby rozwiązać konflikt a oba branche mogły wprowadzić swoje zmiany.
    e. Można wykorzystać 'git stash' aby schować i zabezpieczyć zmiany i przejść na inny branch, wykonać tam pracę, którą mamy do wykonania a następnie wrócić i korzystając 'git stash apply' wrócić do pierwotnej pracy

FRONT:

1. Znam bardzo dobrze Reacta oraz jego pochodne (Next JS / React Native). Pracowałem z nim pracując wcześniej w GG International oraz przez ostatnie półtorej roku tworzę w nim aplikację mobilną hulajSKO wraz z customowym CMS do jej zarządzania.
2. Potrafię korzystać z ESLint jako rozszerzenia do VSCode jak również tworzyć customowe rulesety.
3. Funkcyjne komponenty reacta wykorzystują ES6 więc mam dosyć sporą styczność z jego elementami. Powiedziałbym, że mam dosyć spore pojęcie na temat ich działania jak i sposobów ich wykorzystania.
4. Do tej pory wykorzystywałem głównie JWT(JSON Web Token) w ciasteczkach lub headerze 'Authorization' do autoryzowania klienta (frontend) z backendem.
5. Nie korzystałem nigdy z żadnej automatyzacji.
6. Znam większość funkcji SASS i ich zastosowania.
7. Zawsze starałem się tworzyć wszystkie projekty od zera wykorzystując czysty HTML/CSS(SCSS)/JS lub frameworka typu React/Laravel

BACK:

1. -
2. Znam node i jest to mój go-to backend do większości aplikacji, które tworzyłem. W większości przypadków korzystałem ze szkieletu ExpressJS lub ApolloServer(gdy korzystałem z GraphQL zamiast REST/CRUD). Promise to prekursor do asynchronicznego kodu w JavaScripcie, który pozwala funkcji, która ją wywołuje nadal kontynuować działanie gdy ta kalkuluje i zwraca jakąś wartość gdy skończy. Pozwala to uniknąć korzystania z 'callbacków', które są dużo mniej intuicyjne i produkują brzydszy i mniej zrozumiały kod. W ES7 wprowadzono async/await, który bazuje na Promises lecz pozwala na zredukowanie ilości kodu oraz lepsze radzenie sobie z błędami gdyż błąd w Promise nie zostałby złapany w try/catch bez użycia .catch() co spowodowałoby, że musielibyśmy duplikować nasz kod.
3. Myślę, że najbardziej zaznajomiony jestem z PHP 7 jednak w ostatnim czasie zacząłem korzystać więcej z PHP 8.
4. API
    a. Przy tworzeniu API najczęściej korzystałem z:
        i. 200 - Gdy wszystko poszło zgodnie z planem a klient otrzymuje dane.
        ii. 301 - Gdy nastąpiło przekierowanie do innego adresu.
        iii. 400 - Serwer oczekuje parametrów zapytania od klienta i te są błędne lub niekompletne.
        iv. 401 - Klient nie jest znany serwerowi i ten odmawia przekazania danych.
        v. 403 - Brak uprawnień (klient jest znany jednak nie posiada potrzebnych uprawnień)
        vi. 404 - Serwer nie posiada danych, których chce klient.
    b. Nie korzystałem z tego, jednak posiadam wiedzę na ten temat i byłbym w stanie stosować się do tego w nowych projektach.
    c. Staram się aby endpointy był krótkie, najlepiej jednosłowne i przemawiały za siebie:
        i. GET /todos - Zapytanie o wszystkie todo
        ii. GET /todo/:slug - Zapytanie o konkretne todo
        iii. POST /todo - Dodanie nowego todo
        iv. PUT /todo/:slug - Edycja konkretnego todo
        v. DELETE /todo/:slug - Usunięcie konkretnego todo
    d.
        i. Ograniczanie ładowność zapytań (Paginacja)
        ii. Cachowanie zapytań. (W przypadku jeżeli dane się nie zmieniają często lub gdy zmieniają się regularnie)
        iii. Zastosowanie jakiegoś systemu blokowania ataków DDoS
5. -

METODYKI:

1. Do tej pory z TDD korzystałem jedynie przy serwerze NodeJS. Proces mniej więcej wygląda tak: Uruchamiamy testy jednostkowe. W moim przypadku korzystałem z biblioteki Jest. Wszystkie testy powinny przejść, jest to nasz baseline - linia startowa. Tworzymy nowy test, który wprowadzamy w miejsce gdzie ma pojawić się nowa funkcjonalność, zakładając, że funkcjonalność została wcześniej przemyślana to opieramy tworzony test na wymaganiach, jakie dana funkcjonalność powinna spełnić aby przejść. Następnie tworzymy minimalny kod, który przejdzie dany test i ewentualnie przebudowujemy kod w zależności od potrzeb z założeniem, żeby test cały czas był udany.
2. -
3. - OOP
    a. keyword ‘access modifier’ (pozwala na zabezpieczenie niektórych wartości lub metod przed nieautoryzowanym użyciem)
        i. public - jest domyślny więc używanie przychodzi z łatwością. Dzięki niemu naszą metodę lub wartość można używać wszędzie w całej aplikacji.
        ii. protected - metoda lub wartość może być używana wyłącznie w klasie w której się znajduje lub w klasie, do której ta przynależy.
        iii. private - metoda lub wartość może być używana wyłącznie w klasie w której się znajduje.
    b. keyword ‘final’ (pozwala nam na zablokowanie metody lub wartości przed nadpisaniem.
    c. osobiście posiłkuje się bardziej composition, ze względu na to, że możemy modyfikować kod ‘Parent’ klasie bez potrzeby modyfikowania interfejsów klas podległych.
    d. Według mnie używanie ‘scope’ keywords pozwala na czytelne określenie typu zmiennej. Jednak na przykład korzystanie z keyword ‘global’ może przynieść nieoczekiwane skutki.
    e. Domyślnie w PHP korzysta się z ‘pass_by_value’ więc zwykle się tego trzymałem.
4. -
5. -
6. - Metodyka SOLID według mnie jest bardzo istotna w tworzeniu oprogramowania i to dlaczego:
    a. Tworzenie kodu, którego nie modyfikujemy lecz rozszerzamy co pozwala na zachowanie funkcjonalności i brak potrzeby powrotu do wszystkich pochodnych danej metody i modyfikacje ich kodu po każdej zmianie.
    b. Tworzenie metod, które mają tylko jedno zastosowanie i jedną rolę w kodzie. Co pozwala na łatwy debugging i testowanie, gdyż nie musimy sprawdzać wielu czynników.
7. -

FRAMEWORKI:

1. -

a. Według mnie największą wadą frameworków byłoby spowolnienie aplikacji. Framework to bardzo skomplikowana struktura, która zawiera bardzo dużo kodu, którego nie jesteśmy w stanie zmienić co za tym idzie, jeżeli tworzymy bardzo prostą stronę, która ma być szybka i stabilna to albo powinniśmy korzystać z lekkiego frameworka, który okrojony jest ze zbędnych funkcji lub nie korzystać z żadnego frameworka wcale.
b. Zaletą frameworków jest ich funkcjonalność. Pozwalają one zwykle na bardzo prostą integrację z bazami danych, mają wbudowane własne ORM, posiadają wyodrębnione metody cyklu życia komponentów co sprawia, że mamy nad nimi większą kontrolę, oraz jesteśmy w stanie tworzyć mnie statyczne, bardziej dynamiczne aplikacje. Często też pozwalają na łatwy recykling kodu.

2. -