**Jakarta EE. Aplikacje SPA wykorzystujące REST i JSON**

Autorzy prezentacji:

* Natalia Bidzińska
* Rafał Czajka
* Jan Czerlunczakiewicz
* Patryk Ernest
* Grzegorz Karkowski
* Maciej Kobiec

Jakarta EE to platforma oprogramowania zbudowana jako narzędzie do tworzenia oprogramowania biznesowego. Zapewnia nowe technologie i funkcje, które przyspieszają rozwój i skalowanie aplikacji biznesowych do wdrażania w chmurze. Pierwsza wersja Jakarty pojawiła się w 2019 roku i w bezpośredni sposób nawiązywała do oprogramowania Java EE 8, a także utrzymywała z nią całkowitą wsteczną kompatybilność. Ze względu na komplikacje prawne przy przejęciu platformy przez fundację Eclipse od Oracle, jej nazwa została zmieniona na Jakarta. Wraz z nazwą zmienione zostały również przestrzenie nazw i pakietów, czego skutkiem jest brak kompatybilności Jakarty EE 9 z poprzednimi wersjami.

W praktyce jest to zbiór specyfikacji odpowiedzialnych za m. in. komunikację z bazą danych, wstrzykiwanie zależności czy też łączność sieciową. Poprzez specyfikację należy tutaj rozumieć zbiór interfejsów, klas abstrakcyjnych, czy typów wyliczeniowych, oraz dokument opisujący ich zachowanie. W skład platformy wchodzi wiele specyfikacji, m. in.:

* Servlet – używane w celu nawiązywania połączeń sieciowych,
* ServerPages – czyli silnik szablonów Java stosowany w oprogramowaniu z widokami powstającymi po stronie serwera,
* JSON Processing – odpowiadający za mapowanie elementów do JSON i w przeciwnym kierunku,
* Bean Validation – moduł stworzony do walidacji danych,
* Persistence – przeznaczone do mapowania obiektowo relacyjnego i nawiązywania połączeń z bazami.

**RESTful Server w Jakarta EE**

Platforma Jakarta daje możliwość współpracy z licznymi serwerami, np. Apache Tomcat, Jetty, Glassfish czy Wildfly. W tym przypadku referencyjną implementacją jest Glassfish 6. Serwer ma tworzyć środowisko pozwalające na uruchomienie poszczególnych części składowych platformy i umożliwić połączenie z bazą danych, uwierzytelnianie oraz autoryzowanie informacji, czy też obsługę kolejek komunikatów. W przypadku Jakarty EE 8 serwer domyślnie jest wyposażony we wszystko, co jest konieczne do stworzenia aplikacji typu SPA. W kolejnej części przedstawiony zostanie proces konfiguracji serwera RESTful.

Aby skonfigurować taki serwer, należy wykonać następujące kroki:

1. Utworzenie projektu. Należy wybrać opcję *Maven Project* i w konfiguracji projektu ustawić pola *GroupId* jako *book.jakarta8*, *ArtifactId* jako *restdate* oraz wersję na *0.0.1-SNAPSHOT.*
2. Następnie należy się upewnić, że używany JDK jest w wersji 8.
3. Po tym należy otworzyć plik *pom.xml*. W nim trzeba ustawić odpowiednie zależności i sprawdzić, czy zawartość pliku jest zgodna z przedstawioną na slajdzie. Należy również pamiętać o zaktualizowaniu *Maven’a* po dokonaniu zmian.
4. Kolejnym krokiem jest przekonwertowanie projektu (jeżeli nie zostało to zrobione automatycznie). W tym celu należy wybrać opcję *Configure → Convert to Faceted Form.*
5. W projekcie powinien znajdować się wygenerowany automatycznie folder *src/main/webapp/WEB-INF*, zawierający pliki *glassfish-web.xml*, *beans.xm*l (ten plik powinien być pusty) oraz *web.xml*.
6. Jeśli wszystko zostało poprawnie skonfigurowane, należy utworzyć klasę *RestDate* zgodnie z zaprezentowanym listingiem. Jej zadanie jest proste - zwrócić aktualną datę i godzinę wraz ze strefą czasową.
7. Po tym pozostaje jedynie wystartować serwer, zainstalować na nim utworzoną aplikację, wybierając w menu opcję *Run as → Run on Server* i uruchomić w terminalu polecenie:  
   curl -X GET <http://localhost:8080/restdate/webapi/d>
8. W odpowiedzi powinna zostać zwrócona aktualna data i czas.

W ten prosty sposób udało się utworzyć w pełni funkcjonalny serwer.

**Single Page Application**

SPA to akronim, którego rozwinięcie to Single Page Web Application, czyli aplikacja, która zawarta jest na jednej stronie internetowej, a mówiąc bardziej technicznie - w jednym dokumencie HTML. W momencie wejścia do aplikacji pobierana jest niezbędna treść statyczna oraz skrypty napisane w języku javascript, to one będą odpowiedzialne za dynamiczne doładowywanie i ukrywanie zbędnej zawartości witryny w zależności od akcji wykonanych przez użytkownika systemu. Dzięki takiemu podejściu do tworzenia aplikacji webowych w momencie gdy użytkownik np. chce przejść do jednej z podstron serwisu, serwer nie musi wysyłać po raz kolejny całego dokumentu HTML wraz z niezbędną treścią. Wystarczy wtedy pobrać jedynie wymagane zasoby, a niepotrzebne ukryć lub usunąć ze struktury DOM. Na czas trwania takiej operacji można wykorzystać jeden ze współczesnych konceptów UX/UI dotyczący jak takie oczekiwanie “umilić” użytkownikowi, np. wyświetlić kręcący się spinner lub zastosować wzorzec skeleton. Stosowanie takiego podejścia do wytwarzania oprogramowania jakim są współczesne aplikacje webowe sprawia, że aplikacja jest wydajna oraz relatywnie łatwo skalowalna, a co najważniejsze - płynna i przyjemna w odbiorze dla użytkownika.

Krótko o historii - pierwsze SPA powstawały jako aplikacje Flash i aplety języka Java. Możliwości tworzenia aplikacji webowych zostały zrewolucjonizowane w 2006 roku, kiedy wydany został asynchroniczny JavaScript i XML (AJAX). W niedalekiej przyszłości świat FrontEnd developerów wzbogaci się o javascriptowe frameworki, ułatwiające wytwarzanie coraz to lepszego jakościowo oprogramowania. Nowe możliwości zmieniły również panujące trendy w projektowaniu stron pod kątem UX/UI. Coraz większy nacisk kładzie się na optymalizację procesów oraz komfort i wrażenia użytkownika końcowego. W końcu, SPA stało się najczęściej preferowanym podejściem do wytwarzania aplikacji webowej.

**SPA - popularne biblioteki i frameworki**

SPA po dziś dzień jest najpopularniejszym sposobem tworzenia stron i aplikacji internetowych. Wpływ na to miało powstanie wielu bibliotek i frameworków przeznaczonych do ich budowania. Najpopularniejszymi z nich są:

* React
* Vue
* Angular
* Svelte

Należy jednak pamiętać, że świat JavaScript’owych i TypeScript’owych frameworków szybko się zmienia i popularne obecnie rozwiązania nie muszą takie być za kilka lat. Obecnie powraca moda na tworzenie aplikacji SSR. Popularnym rozwiązaniem jest też SSG (ang. Static Site Generation). Z tego powodu programiści często wybierają nowo powstające frameworki takie jak: Next.js, Nuxt.js, SvelteKit, Remix, które jednak budowane są w większości na bazie wcześniej wymienionych frameworków. Nauka Reacta czy Vue na pewno nie będzie więc czasem straconym.

**REST**

Co najważniejsze, REST nie jest protokołem!! Jest to zbiór dobrych praktyk tworzenia architektury aplikacji rozproszonych (wzorzec). Najczęściej kojarzony jest z HTTP, choć nie jest z nim ściśle związany. Nie jest standardem, lecz zbiorem ograniczeń.

W tabeli przedstawione zostały ogólne zasady REST:

| **Zasoby** | Wyeksponowane przez łatwo zrozumiałe identyfikatory katalogów (URI) |
| --- | --- |
| **Obiekty** | Reprezentowane i przekazywane w postaci JSON |
| **Zapytania** | Wykorzystują jawnie metody HTTP (GET, POST, PUT, DELETE) |
| **Interakcje** | Są całkowicie bezstanowe. Kontekst klienta nie jest przechowywany na serwerze między żądaniami |

**Zasady tworzące REST**

Aby API można nazwać RESTful lub API RESTowym musi ono spełniać kilka założeń:

1. Odseparowanie interfejsu użytkownika od operacji na serwerze. Klient poprzez „wydawanie poleceń” nie ma wpływu na to co się dzieje po stronie serwera. Działa to również w drugą stronę – serwer daje klientowi jedynie odpowiedź i nie ma prawa ingerować w UI. Pozwala to na korzystanie z jednego REST API w wielu niezależnych od siebie aplikacjach, a dane pozostaną spójne.
2. Bezstanowość – mówi się że REST jest stateless – oznacza to, że każde zapytanie od klienta musi zawierać komplet informacji oraz, że serwer nie przechowuje stanu o sesji użytkownika po swojej stronie. W REST nie istnieją takie pojęcia jak chociażby stany czy sesje.
3. Cacheability – odpowiedź, którą użytkownik otrzyma z REST API musi jasno definiować, czy ma ona być cacheable czy non-cacheable. Ma to znaczenie przy danych, które bardzo szybko stają się nieaktualne oraz przy danych, które aktualizują się relatywnie rzadko – nie ma sensu na przykład cache’ować współrzędnych geograficznych pędzącego samolotu, natomiast już jego kolor czy nazwę już tak.
4. Endpointy, czyli adresy zasobów, powinny jednoznacznie wskazywać do jakiego zasobu się odwołują. Z ich budowy powinniśmy wiedzieć jaki konkretnie zasób otrzymamy. Co ważne – dane otrzymywane w API powinny być niezależne w żaden sposób od schematu bazy danych w jakiej są przetrzymywane. Oczywiście nie ma przeciwwskazań, aby struktura danych wyglądała identycznie jak schemat bazy danych – niemniej jednak struktura w żaden sposób nie powinna zależeć od tego schematu.
5. Separacja warstw – powinniśmy oddzielić warstwy dostępu do danych, logiki biznesowej oraz prezentacji. Żadna z warstw nie powinna bezpośrednio oddziaływać na inne warstwy. Użycie (implementacja) pośrednich i zewnętrznych API powinny być ukryte. Przykładem może być wcześniej wspomniany samolot. Dla przykładu, informacja o kolorze może pochodzić z zupełnie innego API – klient nie musi o tym wiedzieć.

**Metody uwierzytelniania w REST API**

W REST najczęściej stosowane metody uwierzytelnienia to:

* basic,
* api-key,
* oauth.

W przypadku metody basic, W nagłówku żądania HTTP jest wysyłane pole authorization basic, jest to prosta metoda uwierzytelniania, która zapewnia minimalny poziom zabezpieczenia API.

W metodzie uwierzytelniania za pomocą APIkey dla klienta jest przydzielany unikalny klucz i na jego podstawie, klient jest autoryzowany. Na przykładzie pokazane jest w jaki sposób wygląda nagłówek w przypadku przekazania APIKey.

W przypadku uwierzytelniania oauth2 klient najpierw musi uzyskać access token z serwera, który następnie używa aby uzyskać dostęp do API.

**Wady i zalety REST**

Jako wadę REST można zaliczyć bezstanowość, powoduje to konieczność implementacji dodatkowych mechanizmów po stronie klienta, ale z drugiej strony zwiększa to skalowalność, można w ten sposób na przykład dodać więcej mikro-serwisów obsługujących ten sam endpoint. Bezstanowość wymusza dodanie często redundantnych informacji. Sam REST jest łatwy w implementacji i umożliwia komunikację niezależnie od języka programowania.

**JSON**

JSON, czyli JavaScript Object Notation, to format zapisu i wymiany danych. Jest to format tekstowy, uważany za czytelny zarówno dla maszyn, jak i dla ludzi. W przypadku przesyłania danych za pomocą protokołu http, jeżeli chcemy w body zamieścić dane w formacie JSON, wówczas należy ustawić odpowiedni nagłówek - *Content-Type: application/json*.

Na slajdzie przedstawiony został prawidłowy dokument w formacie JSON. Jak widać, składa się on z nawiasów klamrowych, wewnątrz których umieszczane są pary w postaci: “nazwa atrybutu”: <wartość atrybutu>. Nazwa atrybutu jest łańcuchem znaków, zaś wartość atrybutu może być dowolną wspieraną wartością.

Przesyłanie danych w formacie JSON ma wiele zalet. Jak już wspomniano, jest on czytelny zarówno dla maszyn, jak i dla ludzi. Dodatkowo chroni przed atakami CSRF, zajmuje mniej miejsca (ponieważ używanych jest mniej znaków niż np. w XML), łatwo jest przedstawić w nim wartość null oraz bool, a także jest łatwy do parsowania.

**Asynchroniczne zapytania do serwera**

Zanim nastał czas aplikacji typu SPA, najpowszechniejszą metodą tworzenia stron internetowych był SSR (ang. Server Side Rendering). Jest to sposób na wygenerowanie pliku HTML. Przeglądarka internetowa wysyła żądanie pobrania strony, która odpowiada wysyłając odpowiedź z kompletnym, wyrenderowanym plikiem HTML.

Sposób tworzenia stron i aplikacji internetowych zaczął się jednak drastycznie zmieniać, a zmiany te zapoczątkowane były między innymi powstaniem technologii Ajax (ang. Asynchronous JavaScript and XML). Technologia ta umożliwia wykonywanie asynchronicznych zapytań do serwera i umożliwia przeładowanie jedynie część strony. JavaScript obecnie posiada natywny sposób do wykonywania takich zapytań o nazwie Fetch API. Bardzo często stosowana jest jednak jego alternatywa, a mianowicie biblioteka Axios. Posiada ona wiele przydatnych funkcjonalności takich jak: interceptory, możliwość anulowania zapytań czy globalną konfigurację.

**Podsumowanie**

Czy każda aplikacja zawarta na jednej stronie to SPA? Chociaż w pierwszej chwili można by pomyśleć, że to prawda, to jednak tak nie jest. Często zdarza się, że SPA mylona jest z witrynami typu “one page”, czyli takimi, które pozbawione są nawigacji oraz podstron, a sama zawartość umieszczona jest w segmentach, do których można ewentualnie scrollować. Single Page Application dodatkowo umożliwia dodawanie podstron, które znajdować się będą pod innym URL’em, co w przypadku rozbudowanych aplikacji ułatwia nawigację.

Stosowanie SPA ma wiele zalet. Z punktu widzenia użytkownika jest to m. in. możliwość wyświetlenia splashscreenów, spinnerów, skeletonów etc. w oczekiwaniu na dane oraz zwiększona wydajność gdy podstawowe dane zostaną załadowane. Z kolei z punktu widzenia dewelopera najważniejsze zalety to separacja warstwy klienckiej od serwerowej, odciążenie serwera, i możliwość migracji na inny typ aplikacji. Aktualnie wiele popularnych serwisów jest zbudowanych w oparciu o SPA. Zaliczają się do nich m. in. Facebook, YouTube, Twitter, GitHub, Gmail i wiele innych.