

Spis treści

Przedmowa (11)

Podziękowania (13)

O książce (15)

O autorze (19)

CZĘŚĆ I POCZĄTEK PRZYGODY Z MIKROFRONTENDAMI (21)

1. Czym jest mikrofrontend? (23)

1.1. Szerszy kontekst (24)

1.1.1. Systemy i zespoły (26)

1.1.2. Frontend (28)

1.1.3. Integracja frontendu (31)

1.1.4. Wspólne tematy (32)

1.2. Jakie problemy rozwiązuje architektura mikrofrontendowa? (34)

1.2.1. Optymalizacja rozwoju funkcjonalności (34)

1.2.2. Precz z frontendowym monolitem (34)

1.2.3. Gotowość na zmiany (36)

1.2.4. Korzyści z niezależności (38)

1.3. Wady architektury mikrofrontendowej (40)

1.3.1. Redundancja (40)

1.3.2. Spójność (41)

1.3.3. Różnorodność technologii (41)

1.3.4. Więcej kodu frontendowego (42)

1.4. Kiedy stosować model mikrofrontendowy? (42)

1.4.1. Dla średnich i dużych projektów (42)

1.4.2. Najlepiej działa w sieci (43)

1.4.3. Produktywność i koszty (44)

1.4.4. Kiedy unikać mikrofrontendów? (44)

1.4.5. Kto używa mikrofrontendów? (45)

1.5. Podsumowanie (46)

2. Mój pierwszy projekt mikrofrontendowy (49)

2.1. Przedstawiamy TraktoryOnline (50)

2.1.1. Zaczynamy (51)

2.1.2. Uruchamianie aplikacji (52)

2.2. Przejścia między stronami za pomocą łączy (54)

2.2.1. Własność danych (54)

2.2.2. Kontrakt między zespołami (55)

2.2.3. Jak to zrobić? (56)

2.2.4. Zmiany adresów URL (59)

2.2.5. Korzyści (59)

2.2.6. Wady (60)

2.2.7. Kiedy stosować łączy? (60)

2.3. Kompozycja za pomocą ramek iframe (61)

2.3.1. Jak to zrobić? (61)

2.3.2. Korzyści (63)

2.3.3. Wady (63)

2.3.4. Kiedy stosować ramki iframe? (64)

2.4. Co dalej? (65)

2.5. Podsumowanie (66)

CZĘŚĆ II ROUTING, KOMPOZYCJA I KOMUNIKACJA (67)

3. Kompozycja techniką AJAX i routing po stronie serwera (69)

3.1. Kompozycja techniką AJAX (70)

3.1.1. Jak to zrobić? (71)

3.1.2. Przestrzenie nazw dla stylów i skryptów (73)

3.1.3. Deklaratywne ładowanie z biblioteką h-include (76)

3.1.4. Korzyści (77)

3.1.5. Wady (78)

- 3.1.6. Kiedy stosować technikę AJAX? (79)
 - 3.1.7. Podsumowanie (79)
- 3.2. Routing przez współdzielony serwer WWW (80)
 - 3.2.1. Jak to zrobić? (82)
 - 3.2.2. Przestrzenie nazw dla zasobów (85)
 - 3.2.3. Metody konfiguracji przekierowań (86)
 - 3.2.4. Odpowiedzialność za infrastrukturę (87)
 - 3.2.5. Kiedy stosować? (88)
- 3.3. Podsumowanie (89)
- 4. Kompozycja po stronie serwera (91)
 - 4.1. Kompozycja - Nginx i SSI (92)
 - 4.1.1. Jak to zrobić? (93)
 - 4.1.2. Szybsze ładowanie (95)
 - 4.2. Obsługa niepewnych fragmentów (97)
 - 4.2.1. Zawodny fragment (97)
 - 4.2.2. Integracja fragmentu "W okolicy" (98)
 - 4.2.3. Limity czasu i treści zastępcze (99)
 - 4.2.4. Treści zastępcze (101)
 - 4.3. Analiza wydajności łączenia fragmentów (102)
 - 4.3.1. Ładowanie równoległe (102)
 - 4.3.2. Zagnieżdżone fragmenty (103)
 - 4.3.3. Opóźnione ładowanie (103)
 - 4.3.4. Czas do odebrania pierwszego bajta a streaming (104)
 - 4.4. Szybki przegląd innych rozwiązań (106)
 - 4.4.1. Edge-Side Includes (106)
 - 4.4.2. Tailor firmy Zalando (107)
 - 4.4.3. Podium (109)
 - 4.4.4. Którego rozwiązania użyć? (115)
 - 4.5. Zalety i wady kompozycji po stronie serwera (115)
 - 4.5.1. Korzyści (115)

- 4.5.2. Wady (116)
 - 4.5.3. Kiedy integracja po stronie serwera ma sens? (117)
- 4.6. Podsumowanie (117)
- 5. Kompozycja po stronie klienta (119)
 - 5.1. Mikrofrontend w komponencie webowym (120)
 - 5.1.1. Jak to zrobić? (122)
 - 5.1.2. Framework w komponencie webowym (126)
 - 5.2. Izolacja stylów dzięki mechanizmowi Shadow DOM (128)
 - 5.2.1. Tworzenie ukrytego elementu głównego (128)
 - 5.2.2. Ograniczenie widoczności stylów (129)
 - 5.2.3. Kiedy używać mechanizmu Shadow DOM? (131)
 - 5.3. Zalety i wady komponentów webowych jako metody kompozycji (132)
 - 5.3.1. Korzyści (132)
 - 5.3.2. Wady (132)
 - 5.3.3. Kiedy integracja po stronie klienta ma sens? (133)
 - 5.4. Podsumowanie (134)
- 6. Wzorce komunikacji (135)
 - 6.1. Komunikacja interfejsów użytkownika (136)
 - 6.1.1. Od komponentu nadrzędnego do podrzędnego (137)
 - 6.1.2. Komunikacja od komponentu podrzędnego do nadrzędnego (140)
 - 6.1.3. Komunikacja między fragmentami (144)
 - 6.1.4. Mechanizm publikacji/subskrypcji interfejsu API Broadcast Channel (148)
 - 6.1.5. Kiedy komunikacja między interfejsami użytkownika się sprawdza? (149)
 - 6.2. Inne mechanizmy komunikacji (150)
 - 6.2.1. Globalny kontekst i autentykacja (151)
 - 6.2.2. Zarządzanie stanem (152)
 - 6.2.3. Komunikacja między frontendem a backendem (153)
 - 6.2.4. Replikacja danych (153)
 - 6.3. Podsumowanie (155)

7. Routing po stronie klienta i powłoka aplikacji (157)

7.1. Powłoka aplikacji z płaskim routingiem (160)

7.1.1. Czym jest powłoka aplikacji? (160)

7.1.2. Anatomia powłoki aplikacji (160)

7.1.3. Routing po stronie klienta (162)

7.1.4. Renderowanie stron (164)

7.1.5. Kontrakty między powłoką aplikacji a zespołami (167)

7.2. Powłoka aplikacji z dwupoziomowym routingiem (168)

7.2.1. Router pierwszego poziomu (169)

7.2.2. Routing na poziomie zespołu (170)

7.2.3. Co się dzieje przy zmianie adresu? (171)

7.2.4. Interfejsy API powłoki aplikacji (174)

7.3. Rzut oka na metaframework single-spa (175)

7.3.1. Jak działa single-spa? (176)

7.4. Wady połączonych aplikacji jednostronicowych (181)

7.4.1. Tematy do przemyślenia (181)

7.4.2. Kiedy stosować połączone aplikacje jednostronicowe? (184)

7.5. Podsumowanie (185)

8. Kompozycja i uniwersalne renderowanie (187)

8.1. Połączenie dwóch kompozycji (188)

8.1.1. Mechanizm SSI i komponenty webowe (190)

8.1.2. Kontrakt między zespołami (194)

8.1.3. Inne rozwiązania (195)

8.2. Kiedy stosować uniwersalną kompozycję? (195)

8.2.1. Uniwersalne renderowanie z kompozycją wyłącznie po stronie serwera (196)

8.2.2. Większa złożoność (196)

8.2.3. Uniwersalnie renderowane połączone aplikacje jednostronicowe? (196)

8.3. Podsumowanie (197)

- 9. Który rodzaj architektury pasuje do mojego projektu? (199)
 - 9.1. Przypomnienie terminologii (200)
 - 9.1.1. Routing i przejścia między stronami (201)
 - 9.1.2. Metody kompozycji (201)
 - 9.1.3. Wysokopoziomowe modele architektoniczne (203)
 - 9.2. Porównanie złożoności (206)
 - 9.2.1. Architektoniczna niejednorodność (207)
 - 9.3. Witryna czy aplikacja? (207)
 - 9.3.1. Kontinuum między dokumentem a aplikacją (208)
 - 9.3.2. Serwer, klient czy uniwersalne renderowanie? (209)
 - 9.4. Wybór odpowiedniej architektury i metody integracji (210)
 - 9.4.1. Silna izolacja (przestarzały kod / kod strony trzeciej) (212)
 - 9.4.2. Szybkie pierwsze załadowanie / stopniowe ulepszanie (212)
 - 9.4.3. Błyskawiczna reakcja (213)
 - 9.4.4. Miękką nawigacja (214)
 - 9.4.5. Wiele mikrofrontendów na jednej stronie (214)
 - 9.5. Podsumowanie (215)

CZĘŚĆ III SZYBKOŚĆ, SPÓJNOŚĆ I EFEKTYWNOŚĆ (217)

- 10. Ładowanie zasobów (219)
 - 10.1. Strategie odnoszenia się do zasobów (220)
 - 10.1.1. Odniesienia bezpośrednie (220)
 - 10.1.2. Cache-busting a niezależne wdrożenia (221)
 - 10.1.3. Odniesienie przez przekierowanie (klient) (222)
 - 10.1.4. Odniesienia w dyrektywie include (225)
 - 10.1.5. Synchronizacja wersji kodu i zasobów (227)
 - 10.1.6. Zasoby we fragmencie (230)
 - 10.1.7. Zintegrowane rozwiązania (Tailor, Podium itp.) (230)
 - 10.1.8. Szybkie podsumowanie (232)
 - 10.2. Podział na pakiety (233)

- 10.2.1. Protokół HTTP/2 (233)
- 10.2.2. Wszystko w jednym pakiecie (234)
- 10.2.3. Pakiet na zespół (235)
- 10.2.4. Pakiet dla każdej strony i fragmentu (235)
- 10.3. Ładowanie na żądanie (235)
 - 10.3.1. Komponenty pośredniczące (236)
 - 10.3.2. Opóźnione ładowanie stylów (237)
- 10.4. Podsumowanie (237)
- 11. Wydajność to klucz (239)
 - 11.1. Projektowanie z myślą o wydajności (240)
 - 11.1.1. Różne zespoły, różne pomiary (240)
 - 11.1.2. Międzyzespołowy budżet wydajności (242)
 - 11.1.3. Odpowiedzialność za spowolnienia (243)
 - 11.1.4. Korzyści w obszarze wydajności (244)
 - 11.2. Biblioteki zewnętrzne (246)
 - 11.2.1. Koszt autonomii (246)
 - 11.2.2. Małe jest piękne (247)
 - 11.2.3. Jedna globalna wersja (249)
 - 11.2.4. Wersjonowane pakiety z bibliotekami zewnętrznymi (250)
 - 11.2.5. Unikaj współdzielenia kodu biznesowego (262)
 - 11.3. Podsumowanie (262)
- 12. Interfejs użytkownika i system projektowania (265)
 - 12.1. Po co nam system projektowania? (266)
 - 12.1.1. Cel i rola (268)
 - 12.1.2. Korzyści (268)
 - 12.2. Centralny system projektowania a autonomia zespołów (269)
 - 12.2.1. Czy potrzebuję własnego systemu projektowania? (269)
 - 12.2.2. Proces, a nie projekt (270)
 - 12.2.3. Stały budżet i odpowiedzialność (270)

- 12.2.4. Poparcie zespołów (271)
- 12.2.5. Proces rozwoju - centralny czy federacyjny? (273)
- 12.2.6. Etapy rozwoju (274)
- 12.3. Integracja w czasie wykonania czy wersjonowanie? (276)
 - 12.3.1. Integracja w czasie wykonania (276)
 - 12.3.2. Wersjonowany pakiet (278)
- 12.4. Artefakty generyczne i specyficzne (281)
 - 12.4.1. Wybór formatu komponentów (281)
 - 12.4.2. Nieuchronne zmiany (285)
- 12.5. Co powinno wchodzić w skład centralnej biblioteki wzorców? (286)
 - 12.5.1. Koszt współdzielenia komponentów (286)
 - 12.5.2. Centralny czy lokalny? (286)
 - 12.5.3. Centralne i lokalne biblioteki wzorców (289)
- 12.6. Podsumowanie (290)
- 13. Zespoły i granice (293)
 - 13.1. Dopasowanie między systemami i zespołami (294)
 - 13.1.1. Granice między zespołami (295)
 - 13.1.2. Głębina integracji (297)
 - 13.1.3. Zmiana kulturowa (300)
 - 13.2. Dzielenie się wiedzą (301)
 - 13.2.1. Wspólnota praktyków (302)
 - 13.2.2. Nauka (303)
 - 13.2.3. Zaprezentuj swoją pracę (303)
 - 13.3. Globalne problemy (303)
 - 13.3.1. Centralna infrastruktura (304)
 - 13.3.2. Zespół ds. specjalistycznych komponentów (305)
 - 13.3.3. Globalne uzgodnienia i konwencje (305)
 - 13.4. Różnorodność technologiczna (306)
 - 13.4.1. Zestawy narzędzi i opcje domyślne (306)
 - 13.4.2. Szablon frontendowy (306)

13.4.3. Odrzuć strach przed kopiowaniem (308)

13.4.4. Wartość podobieństw (308)

13.5. Podsumowanie (309)

14. Migracje, lokalne środowisko rozwojowe i testowanie (311)

14.1. Migracja (312)

14.1.1. Model koncepcyjny na przetarcie szlaku (312)

14.1.2. Strategia nr 1: kawałek po kawałku (314)

14.1.3. Strategia nr 2: najpierw frontend (315)

14.1.4. Strategia nr 3: od zera do wielkiego wybuchu (317)

14.2. Rozwój w środowisku lokalnym (318)

14.2.1. Bez kodu pozostałych zespołów (319)

14.2.2. Tworzenie atrap fragmentów (319)

14.2.3. Fragmenty w izolacji (321)

14.2.4. Pobieranie mikrofrontendów innych zespołów ze środowiska testowego lub produkcji (323)

14.3. Testowanie (323)

14.4. Podsumowanie (325)