МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ІНЖЕНЕРНИЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІМ. Ю.М. ПОТЕБНІ
ЗАПОРІЗЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ

КАФЕДРА ЕЛЕКТРОНІКИ, ІНФОРМАЦІЙНИХ СИСТЕМ, ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Кваліфікаційна робота

другий (магістерський)
(рівень вищої освіти)

на тему **Аналіз та дослідження мікрофронтендних архітектур на основі розробки високонавантажених веб-застосунків**

Виконав: студент 2 курсу, групи 8.1212-іпз-2
спеціальності 121 Інженерія програмного забезпечення
(код і назва спеціальності)
освітньої програми Інженерія програмног забезпечення
(код і назва освітньої програми)
В.А. Лазарєв
(ініціали та прізвище)
Керівник доцент, к.ф.-м.н., В.І.Попівщий
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)
Рецензент директор ТОВ «Алтер Віжн Груп»
В.С. Тряпічко
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)

Запоріжжя
2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ІНЖЕНЕРНИЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІМ. Ю.М. ПОТЕБНІ
ЗАПОРІЗЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ

Кафедра електронікки, інформаційних систем та програмного забезпечення
Рівень вищої освіти _____другий (магістерський)_____
Спеціальність ___121_Інженерія програмного забезпечення_____
(код та назва)
Освітня програма ___Інженерія програмного забезпечення
(код та назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри_____ПІП_____
"_01_" __вересня_____2023_року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Лазарєву Вадиму Андрійовичу
(прізвище, ім'я, по батькові)

1.Тема роботи _____Аналіз та дослідження мікрофронтендних архітектур на основі розробки високонавантажених веб-застосунків_____

керівник роботи доцент, к.ф.-м.н Попівщий Василь Іванович
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом ЗНУ від 09.10.2023, № 1577- с_____

2. Строк подання студентом кваліфікаційної роботи__30.12.2023 р.___
3. Вихідні дані магістерської роботи
- комплект нормативних документів ;
- технічне завдання до роботи.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
- огляд та збір літератури стосовно теми кваліфікаційної роботи;
- огляд та аналіз існуючих рішень та аналогів;
- дослідження проблеми розпізнавання мов та розробка методів її вирішення;
- створення програмного продукту та його опис;
- перелік вимог для роботи програми;
- дослідження поставленої проблеми та розробка висновків та пропозицій.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
26 слайдів презентації

6. Консультанти розділів магістерської роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата
		Завдання прийняв

7. Дата видачі завдання 01.09.2023

КАЛЕНДАРНИЙ ПЛАН

з/п	Назва етапів магістерської роботи	Строк виконання етапів магістерської роботи	Примітка
	Аналіз предметної області	02.09-10.09.23	виконано
	Формулювання основної задачі дипломної роботи та узгодження її з науковим керівником	11.09-12.09.23	виконано
	Аналіз існуючих методів рішення	13.09-14.09.23	виконано
	Дослідження архітектур та фреймворків мікрофронтендів	15.09-20.09.23	виконано
	Дослідження засобів розробки мікрофронтедних архітектур	21.09-26.09.23	виконано
	Узгодження подальших дій з науковим керівником	27.09-28.09.23	виконано
	Дослідження викликів між монолітними та мікрофронтендними архітектурами	29.09-13.10.23	виконано
	Розробка додатку з використанням мікрофронтендних архітектур	14.10-16.10.23	виконано
	Представлення отриманих результатів науковому керівнику та узгодження плану подальшого дослідження	17.10-19.10.23	виконано
0	Оформлення звіту	23.11-27.11.23	виконано

Студент _____ Лазарєв В.А
(підпис) (прізвище та ініціали)

Керівник роботи _____ Попівщий В.І.
(підпис) (прізвище та ініціали)

Нормоконтроль пройден

Нормоконтролер _____ Скрипник І.А.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Сторінок: 138

Рисунків: 45

Таблиць: 5

Джерел: 18

Лазарєв В. А. Аналіз та дослідження мікрофронтендних архітектур на основі розробки високонавантажених веб-застосунків : кваліфікаційна робота магістра спеціальності 121 «Інженерія програмного забезпечення» / наук. керівник В. І. Попівший. Запоріжжя : ЗНУ, 2023. 138 с.

Мета дослідження полягає у вивченні основних принципів, переваг та недоліків архітектури мікрофронтендів. Основні завдання включають аналіз сучасних підходів до створення веб-додатків, дослідження технічних аспектів мікрофронтендів та їх практичне застосування.

У процесі дослідження було розглянуто проблему створення та управління масштабованими веб-додатками з використанням концепції мікрофронтендів. Особлива увага була приділена методам розбиття великих фронтенд-проектів на менші, незалежні частини, які можуть бути розроблені, тестовані та випущені окремо одна від одної. Як результат, було розроблено архітектурну модель мікрофронтендів, яка інтегрує різні фреймворки та бібліотеки, такі як React, Vue.js та Angular, у єдиний користувацький інтерфейс. Окрім цього, було створено демонстраційний веб-додаток, що використовує цю модель для підвищення гнучкості та швидкості розробки, а також для ефективного розподілу ресурсів команди розробників.

Ключові слова: Мікрофронтенд, React, Personio, DataDog, Split IO, Next JS, Оркестратор, масштабування, гнучкість, розгортання, CI/CD

SUMMARY

Pages: 138

Figures: 4

Tables: 5

Sources: 18

Lazariev V. A. Analysis and Research of Microfrontend Architectures Based on the Development of High-Load Web Applications: Master's Thesis in Specialty 121 "Software Engineering" / Scientific Supervisor Vasyl Popivshchyi. Zaporizhzhia: ZNU, 2023. 138 p.

The aim of the study is to investigate the main principles, advantages, and disadvantages of microfrontend architecture. The main tasks include the analysis of modern approaches to web application development, research of the technical aspects of microfrontends, and their practical application.

In the course of the research, the problem of creating and managing scalable web applications using the concept of microfrontends was considered. Special attention was given to methods of dividing large frontend projects into smaller, independent parts that can be developed, tested, and released separately. As a result, an architectural model of microfrontends was developed, integrating various frameworks and libraries, such as React, Vue.js, and Angular, into a single user interface. In addition, a demonstration web application was created that uses this model to increase flexibility and development speed, as well as to effectively distribute the resources of the development team.

Keywords: Microfrontend, React, Personio, DataDog, Split IO, Next JS, Orchestrator, scalability, flexibility, deployment, CI/CD.

ЗМІСТ

ВСТУП

Актуальність теми

Розвиток веб-технологій і швидке впровадження сучасних архітектурних рішень змушують розробників шукати нові підходи до створення великих та складних веб-додатків. Мікрофронтенди являють собою один із таких підходів, який дозволяє розбити фронтенд-частину додатку на незалежні від себе модулі. Цей підхід має потенційно велике практичне застосування, оскільки він може полегшити процес розробки, тестування та впровадження веб-додатків.

Мета і завдання дослідження

Мета дослідження полягає у вивченні основних принципів, переваг та недоліків архітектури мікрофронтендів. Основні завдання включають аналіз сучасних підходів до створення веб-додатків, дослідження технічних аспектів мікрофронтендів та їх практичне застосування.

Об’єкт дослідження

Об’єктом дослідження є мікрофронтенди як архітектурне рішення у сфері веб-розробки.

Предмет дослідження

Предметом дослідження є основні принципи роботи, технічні аспекти та практичне застосування мікрофронтендів у сучасних веб-додатках.

Методи дослідження

Для вирішення поставлених завдань використовуються такі методи дослідження:

1. Аналіз та вивчення сучасних підходів до веб-розробки.
2. Вивчення основних принципів роботи мікрофронтендів.
3. Аналіз переваг та недоліків архітектури мікрофронтендів.
4. Практичне застосування та експериментування з мікрофронтендами.

Наукова новизна одержаних результатів

Наукова новизна дослідження полягає в глибокому аналізі та вивченні мікрофронтендів як сучасного підходу до розробки веб-додатків, а також у розробці методології застосування цього підходу в практичних проектах.

Практичне значення одержаних результатів

Практичне значення результатів полягає у можливості використання мікрофронтендів для поліпшення процесу розробки, тестування та впровадження великих і складних веб-додатків, а також у покращенні продуктивності роботи команди розробників.

Апробація одержаних результатів

Результати дослідження були представлені на XVI науково-практичній конференції студентів, аспірантів, докторантів і молодих вчених Запорізького національного університету «Молода наука-2023», а також на XXVIII науково-технічній конференції студентів, аспірантів, магістрантів і викладачів Інженерного навчально-наукового інституту Запорізького національного університету.

Глосарій

Мікрофронтенд (англ. Micro frontend) - це підхід до розробки веб-додатків, при якому фронтенд розбивається на незалежні від себе модулі, кожен з яких відповідає за певний функціонал.

Фронтенд (англ. Frontend) - це частина веб-додатку, яка відповідає за взаємодію з користувачем та відображення інтерфейсу.

Оркестратор - це додаток на базі Next.js, відповідальний за створення основного макету сторінки за моделлю AppShell.

Композиція мікрофронтендів: Цей процес включає динамічне компонування контенту на сторінці, що базується на бізнес-логіці, конфігурації або користувацьких налаштуваннях, на відміну від традиційних односторінкових застосунків, де весь контент завантажується одразу.

Nx - це набір інструментів, що спрощують процес розробки та тестування програмного забезпечення. Він забезпечує ефективне управління залежностями, організацію коду в монорепозиторії та інтеграцію із різними фреймворками та бібліотеками.

Split.io - це платформа для проведення A/B тестування та feature flagging. Вона дозволяє розробникам експериментувати з функціональними можливостями програмного забезпечення, контролюючи доступ до певних функцій серед різних груп користувачів.

Datadog - це сервіс моніторингу та аналітики, який забезпечує централізоване спостереження за додатками, сервісами, інфраструктурою та логами. Datadog надає інструменти для відстеження продуктивності, виявлення збоїв та оптимізації ресурсів.

Jest - це фреймворк для тестування JavaScript, який використовується в основному для тестування фронтенду. Він популярний завдяки своїй простоті у використанні, швидкості та гнучкості у написанні тестів.

Сінтетичні тести - це метод тестування, при якому імітуються дії користувачів або системи для перевірки різних аспектів додатків. Це включає тестування продуктивності, відповіді сервера та інших ключових функцій.

TDD (Test-Driven Development) - це підхід у розробці програмного забезпечення, який полягає в написанні тестів до реалізації функціональності. Спочатку створюються тести для нового коду, потім код пишеться так, щоб він відповідав цим тестам.

Юніт тести - це тип тестування, який фокусується на індивідуальних компонентах або "одиницях" коду. Метою є перевірити, що кожна ізольована частина додатку працює коректно. Юніт тестування є важливою частиною процесу розробки, оскільки воно допомагає ідентифікувати та виправляти помилки на ранніх етапах.

Pipeline у контексті розробки програмного забезпечення відноситься до автоматизованого процесу, через який проходить код або програмне забезпечення від розробки до розгортання. Цей процес зазвичай включає кілька етапів, таких як компіляція коду, автоматизоване тестування, інтеграція та, врешті-решт, розгортання на продуктивному середовищі.

Розгортання - це процес публікації або активації програмного забезпечення у виробничому середовищі. Це останній етап в життєвому циклі розробки програмного забезпечення, під час якого програмний продукт стає доступним для кінцевих користувачів.

GitLab - це веб-платформа для управління репозиторіями коду на базі Git. Вона забезпечує інструменти для співпраці в команді, включаючи відстеження змін, управління завданнями та коментарі до коду. GitLab також пропонує вбудовані функції для неперервної інтеграції та неперервного розгортання (CI/CD).

CI/CD (Continuous Integration/Continuous Deployment) - це практика в розробці програмного забезпечення, яка зосереджується на частій автоматизації

інтеграції коду з основною гілкою репозиторія (Continuous Integration) та його частому автоматичному розгортанні в продуктивному середовищі (Continuous Deployment). Ця практика спрямована на зменшення часу від розробки до використання коду, підвищення якості продукту та забезпечення більш ефективного циклу розробки.

Управління станом у мікрофронтендах - це стосується ситуацій, коли потрібен спільний стан між різними мікрофронтендами, що може вимагати використання глобальних систем управління станом або спеціальних шаблонів для спілкування між ними.

Безпека та ізоляція у мікрофронтендах - це означає забезпечення того, що мікрофронтенди, розроблені різними командами або організаціями, не можуть неправильно взаємодіяти або впливати на інші частини системи.

Респонсивний дизайн у мікрофронтендах - важливість адаптації мікрофронтендів до різних типів пристроїв і розмірів екранів, щоб забезпечити оптимальне відображення і взаємодію користувача з застосунком.

Тестування та надійність мікрофронтендів - включає незалежне тестування кожного мікрофронтенда та комплексне тестування їх інтеграції у реальному застосунку для забезпечення надійності рендерингу.

Масштабування та оптимізація в мікрофронтендах - потреба в оптимізації рендерингу з ростом складності застосунку та кількості користувачів, використання технік, таких як віртуалізація DOM, оптимізація взаємодії з сервером та використання CDN.

РОЗДІЛ 1 ТЕОРЕТИЧНІ АСПЕКТИ МІКРОФРОНТЕНДНИХ АРХІТЕКТУР

1.1 Визначення та класифікація мікрофронтендів

В сучасному світі веб-розробки інновації не припиняються. З ростом інтернету та великої кількості веб-застосунків, потреба в гнучких, масштабованих та ефективних рішеннях лише зростає. У відповідь на ці виклики, індустрія постійно шукає нові методи та підходи до розробки. Однією з таких новаторських концепцій є мікрофронтенди. Мікрофронтенди — це підхід до архітектури веб-застосунків, який пропонує розбивати інтерфейс користувача на менші, незалежні частини.

Ці частини розробляються, [1] тестуються та розгортаються незалежно одна від одної, що дозволяє командам працювати паралельно без конфліктів. Це схоже на підхід мікросервісів у бекенд-розробці, де кожна частина системи розробляється як окремий сервіс. Основна мета мікрофронтендів - це підвищення продуктивності, гнучкості та надійності в процесі розробки. Ця концепція дозволяє командам сконцентруватися на конкретних функціональних частинах застосунку без потреби перейматися іншими частинами. Кожен мікрофронтенд може мати свій власний життєвий цикл, від проектування до розгортання, що полегшує процес оновлення та внесення змін. Основні характеристики мікрофронтендів: Незалежність: Кожен мікрофронтенд є незалежним застосунком, який може мати власний життєвий цикл, від розробки до розгортання. Технологічна гнучкість: Різні мікрофронтенди можуть використовувати різні технології, що дозволяє командам вибирати найкращий стек для конкретного завдання. Ефективна комунікація: Мікрофронтенди можуть спілкуватися між собою за допомогою API, подій чи інших механізмів, але завжди з мінімальними залежностями. Масштабування: Завдяки невеликому

розміру та незалежності, мікрофронтенди легше масштабувати, адаптуючи до змінних вимог бізнесу та користувачів.

Ізоляція: Проблеми в одному мікрофронтенді не впливають на роботу інших, що забезпечує стабільність загального застосунку. Підходи, які лежать в основі мікрофронтендів, були запозичені від мікросервісів — архітектурного стилю для бекенду.

Таким чином, мікрофронтенди можна розглядати як "мікросервіси для фронтенду", де основна ідея полягає в тому, щоб розділити великий застосунок на менші, легко керовані частини. Окрім того, мікрофронтенди відкривають двері для більш гнучкого вибору технологій. Різні команди можуть вибирати різні технологічні стеки для своїх частин застосунку, незалежно від інших частин. Це може бути особливо корисно, коли розглядається інтеграція новітніх технологій або когда різні частини застосунку мають різні вимоги до продуктивності та функціональності.

Важливо зрозуміти, що мікрофронтенди - це не лише технічний підхід, а й організаційний. Вони можуть суттєво вплинути на структуру команд, процеси розробки та способи доставки продуктів кінцевим користувачам. Класифікація мікрофронтендів: Мікрофронтенди можна класифікувати за рядом параметрів: За областю відповідальності: Функціональні мікрофронтенди: Фокусуються на конкретному функціоналі або модулі додатку, такому як авторизація користувача або корзина для покупок. Доменні мікрофронтенди: Фокусуються на конкретному домені або бізнес-області. Наприклад, розділ сайту для управління замовленнями. За методом інтеграції: Зібрані на сервері: Сервер генерує відповідь із зібраними мікрофронтендами і надсилає готовий відгук до клієнта.

Клієнтська інтеграція: Кожен мікрофронтенд завантажується окремо на клієнтській стороні та інтегрується з іншими частинами на клієнті. За способом розгортання: Однакова конфігурація: Всі мікрофронтенди розгортаються на однаковій інфраструктурі з однаковою конфігурацією. Незалежне розгортання:

Кожен мікрофронтенд може мати власні параметри розгортання і може бути розгорнутим окремо. Ця класифікація дозволяє глибше зрозуміти мікрофронтенди та їх різновиди, що сприяє правильному вибору підходу для конкретного проекту.

1.2 Основні принципи роботи мікрофронтендів

Автономність команд:

[1] Мікрофронтенди розроблені так, щоб надати командам максимальну автономність. Кожна команда відповідає за свій власний життєвий цикл продукту: від проектування до розгортання. Це означає, що команди можуть працювати паралельно, приймати рішення без необхідності координації з іншими командами і робити випуски так часто, як це необхідно.

Незалежність розгортання:

Один з основних принципів мікрофронтендів - можливість розгортання частини додатка незалежно від інших частин. Це може значно скоротити час випуску нових функцій та поліпшити продуктивність команди.

Технологічна гнучкість:

Мікрофронтенди не прив'язані до конкретного стеку технологій. Кожна команда може вибирати технології, які найкраще підходять для їх завдань. Це може бути корисно у великих організаціях, де різні команди можуть мати різний досвід і експертизу.

Ізоляція:

Ізоляція гарантує, що помилки або збої в одному мікрофронтенді не вплинуть на роботу інших. Це створює більш стабільний та надійний користувацький досвід.

Композиція:

Мікрофронтенди можуть бути динамічно компоновані на сторінці, щоб створити цілісний користувацький інтерфейс. Це дозволяє додавати, видаляти або замінювати компоненти без переробки всього додатку.

Перевикористання:

Додаткова ізоляція та модульність мікрофронтендів сприяють перевикористання коду. Компоненти можуть бути використані в різних частинах додатку або навіть у різних проектах. Незважаючи на архітектурні розбиття, користувацький досвід повинен залишатися пріоритетом. Інтеграція між мікрофронтендами повинна бути гладкою, щоб користувачі не відчували розривів у взаємодії.

1.3 Історія виникнення та розвитку мікрофронтендів

Мікрофронтенди являють собою відносно новий підхід до розробки веб-застосунків, але коріння їхньої концепції можна простежити до ранніх етапів розвитку веб-розробки. Початки (2000-2010 роки): На ранніх етапах розвитку вебу більшість сайтів була статичними. Однак з появою AJAX та динамічних веб-застосунків розробники почали шукати способи зробити застосунки більш модульними та легко масштабованими. Це було досить викликаючим у світі, де панували монолітні архітектури.

Введення мікросервісів (2010-2015 роки): З розвитком обчислювальної хмари та появою мікросервісів для бекенду, команди почали розглядати можливості розбиття своїх великих застосунків на менші сервіси. Це дозволило їм працювати незалежно, швидко вносити зміни та масштабувати застосунки за потребою.

Поява мікрофронтендів (після 2015 року): Вдосконалюючи підхід мікросервісів на бекенді, розробники фронтенду почали розглядати можливості застосування подібних принципів для інтерфейсів користувача. Це призвело до появи концепції мікрофронтендів. Основна ідея полягала в тому, щоб розділити

великий і складний фронтенд застосунку на менші, незалежні частини. З того часу мікрофронтенди стали все більш популярними, особливо у великих організаціях та на складних проєктах, де команди шукали шляхи оптимізації процесів розробки, тестування та розгортання.

Інтеграція мікрофронтендів (2016-2018 роки): Після введення концепції мікрофронтендів, основний виклик для розробників полягав у тому, як інтегрувати ці незалежні частини в єдиний застосунок. Це стало особливо актуальним, коли в команді було багато мікрофронтендів, які використовували різні технології та версії бібліотек. Різні підходи до інтеграції були запропоновані, включаючи використання веб-компонентів, JavaScript-модулів або навіть серверної інтеграції.

Сучасні тренди та виклики (2019-до сьогодні): З розвитком мікрофронтендів, нові виклики та можливості продовжують з'являтися. Зокрема, ізоляція стилів, управління станом та оптимізація продуктивності стали ключовими темами обговорення в спільноті розробників. Також дедалі більше організацій розглядають можливості гібридних підходів, комбінуючи мікрофронтенди з традиційними архітектурами для досягнення найкращих результатів. З іншого боку, мікрофронтенди допомагають розробникам реагувати на змінні вимоги бізнесу швидше, адаптуючи окремі частини застосунку без потреби великих змін у всьому проекті. Ця гнучкість стає все більш цінною в умовах швидко змінюваного ринку

1.4 Визначення та характеристики мікрофронтендів

Мікрофронтенди з'явилися як відповідь на зростаючу складність сучасних веб-застосунків. Як і інші підходи до декомпозиції програмного забезпечення, такі як мікросервіси, мета мікрофронтендів - це створення більш управлінних, модульних систем.

Визначення: Мікрофронтенд - це підхід до розробки веб-інтерфейсів, де фронтенд-частина додатку розбивається на менші, самостійні частини, які можуть розроблятися і розгортатися незалежно.

Характеристики: Незалежність: Кожен мікрофронтенд може розроблятися, тестуватися та розгортатися незалежно від інших частин системи. Гнучкість: Мікрофронтенди дозволяють командам вибирати технології, які найкраще підходять для їхніх специфічних потреб. Централізоване управління стає менш критичним, оскільки кожен модуль може мати свої власні відповідальності та залежності.

В даній таблиці представлені основні переваги та недоліки мікрофронтендів, які слід враховувати при проектуванні та реалізації.

Таблиця 1

Переваги та недоліки мікрофронтендів

Переваги	Недоліки
Незалежність розробки	Складність управління залежностями
Гнучкість в технологіях	Потенційні проблеми з продуктивністю
Модульність	Вимоги до координації між командами

Виклики. Деякі з основних викликів, які мікрофронтенди намагаються вирішити: Складність управління: Як зростає кодова база, управління нею стає

все більш складним. Узгодження роботи команд: Зі збільшенням кількості команд, потреба в їхньому узгодженні також зростає.

Залежності: Великі системи часто мають численні взаємозалежності, що може призвести до проблем під час розгортання або оновлення.

Технічні аспекти Мікрофронтенди не лише змінюють спосіб, яким команди думають про архітектуру, але і пропонують нові технічні рішення. Вони дозволяють розробникам ізольовано працювати над різними частинами системи без впливу на інші частини.

Таблиця надає загальний огляд технічних аспектів мікрофронтендів, розглядаючи ключові компоненти та їх взаємодію.

Таблиця 2

Технічні аспекти мікрофронтендів

Аспект	Опис
Технологічна гнучкість	Команди можуть вибирати власний стек технологій для своїх модулів.
Ізольоване тестування	Модулі можуть тестуватися окремо, забезпечуючи надійність коду.
Незалежне розгортання	Зміни в одному модулі не вимагають розгортання всього застосунку.

Комунікація між модулями. Одним з ключових питань при роботі з мікрофронтендами є комунікація між різними модулями. Залежно від

архітектури та потреб додатку, може бути використано різні підходи: від простого обміну даними через глобальний стан до використання подій або спеціалізованих шин даних.

Майбутнє мікрофронтендів. Оскільки технологія ще досить нова, спільнота розробників активно досліджує найкращі практики та стратегії для мікрофронтендів. Очікується, що з ростом популярності цього підходу буде більше інструментів, бібліотек та фреймворків, спрямованих на спрощення розробки та інтеграції мікрофронтендів.

1.5 Мікрофронтенди vs Монолітні архітектур

В розробці програмного забезпечення архітектурний підхід відіграє вирішальну роль. Вибір між монолітною архітектурою та мікрофронтендами може вплинути на продуктивність команди, швидкість доставки, а також на здатність системи адаптуватися до змін.

Монолітна архітектура - це підхід до розробки програмного забезпечення, де програма розробляється як єдиний неподільний блок. Всі компоненти системи зазвичай розробляються і тестуються разом, і зміни в одній частині системи часто вимагають повного перегляду всієї системи.

Між монолітами та мікрофронтендами існують ряд ключових відмінностей:

Масштабування: У монолітах масштабування досягається шляхом копіювання всього застосунку на кожен новий сервер, тоді як мікрофронтенди дозволяють масштабувати кожний компонент незалежно. Розробка: В монолітах команди розробників часто працюють над однією кодовою базою, що може призводити до конфліктів і залежностей. У мікрофронтендах кожна команда може працювати незалежно. Технологічний стек: Монолiти часто обмежені одним технологічним стеком, тоді як мікрофронтенди дозволяють використовувати різні технології для різних частин застосунку.

Таблиця 3

Переваги та недоліки мікрофронтендів і монолітів

Аспект	Мікрофронтенди	Моноліти
Масштабування	Горизонтальне, незалежне	Вертикальне, залежне
Розробка	Розподілена, незалежна	Централізована, залежна
Технологічний стек	Гнучкий, може бути різним для кожного модуля	Часто єдиний для всього застосунку
Складність	Вища на початкових етапах, менша при рості	Вища при рості кодової бази
Швидкість розгортання	Швидше, завдяки незалежності модулів	Повільніше, завдяки розміру кодової бази

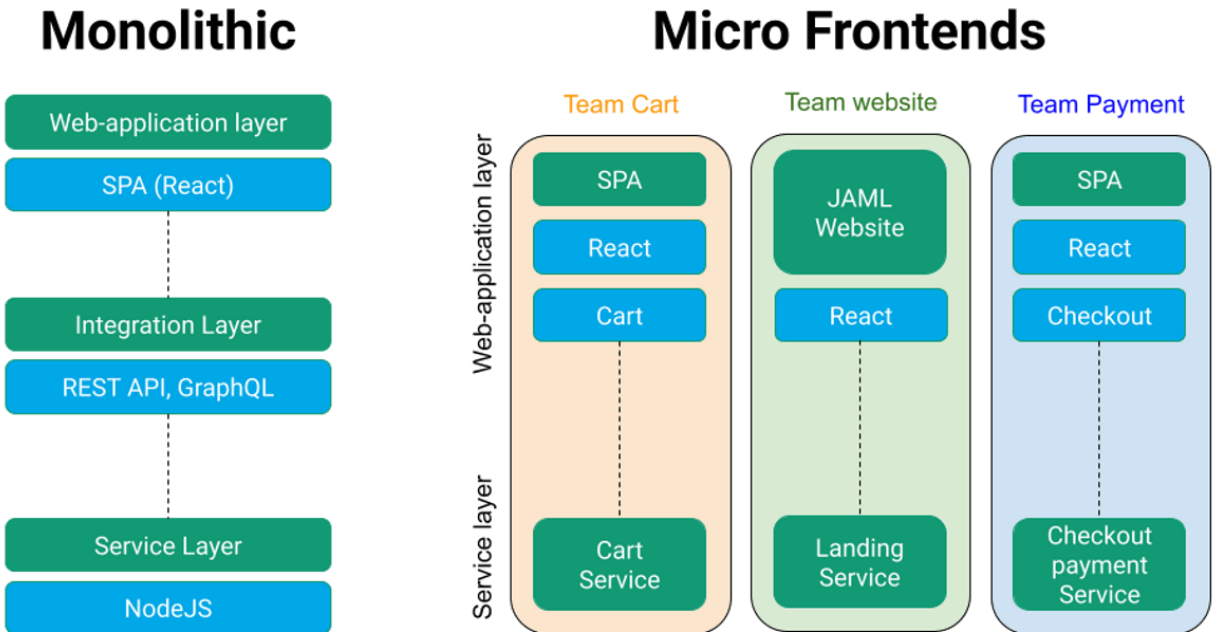

Рисунок — 1 Схеми монолітних веб застосункі/схема мікрофронтендів

1,200 × 627

Мікрофронтенди зазвичай вибирають для великих, розподілених систем, де різні команди можуть працювати над різними частинами застосунку. Наприклад, електронна комерція чи соціальні мережі. Моноліти ж часто є вибором для менших застосунків або стартапів, де швидкість розробки і доставка продукту є ключовими.Впровадження мікрофронтендів або монолітної архітектури в реальних проектах вимагає глибокого розуміння специфіки бізнесу, вимог до продукту та командної динаміки.

Мікрофронтенди: Командна структура: Мікрофронтенди найкраще підходять для великих організацій з множинними командами розробників, де кожна команда може відповідати за свою частину застосунку. Технічні вимоги: Завдяки можливості використовувати різні технологічні стеки, команди можуть вибирати найкращі інструменти для своїх конкретних завдань.Інтеграція: Хоча кожен мікрофронтенд є самостійним, важливо забезпечити гладку інтеграцію між різними частинами застосунку, що може вимагати додаткового коду та координації.

Моноліти: Командна структура: Моноліти часто використовуються в стартапах або менших командах, де швидкість розробки та спростування архітектури є важливими.Технічні вимоги: Оскільки весь застосунок розробляється як єдиний блок, технологічний стек часто є стандартизованим, що може обмежити гнучкість, але спростити розробку. Масштабування: Якщо застосунок росте, монолітичні системи можуть стати складними в управлінні та масштабуванні.

Висновок: Обидва підходи мають свої переваги та недоліки. Вибір між мікрофронтендами та монолітною архітектурою в більшій мірі залежить від конкретних вимог до проекту, командної динаміки та бізнес-цілей. Важливо розуміти, що немає "одного рішення для всіх", і вибір архітектурного підходу повинен базуватися на глибокому аналізі потреб та можливостей конкретного проекту.

1.6 Технічні аспекти мікрофронтендів

Основна ідея мікрофронтендів полягає в тому, щоб дати можливість кожній команді вибирати технологію, яка найкраще підходить для їхньої конкретної частини застосунку. Це може включати в себе: JavaScript-фреймворки: Різні частини застосунку можуть використовувати різні фреймворки, такі як [15] React, Vue, Angular тощо. Стилі: Кожна команда може вибрати свою систему стилів, використовуючи CSS, SASS, LESS або інші. Бібліотеки: За потребою можна інтегрувати зовнішні бібліотеки для анімації, роботи з даними та ін.

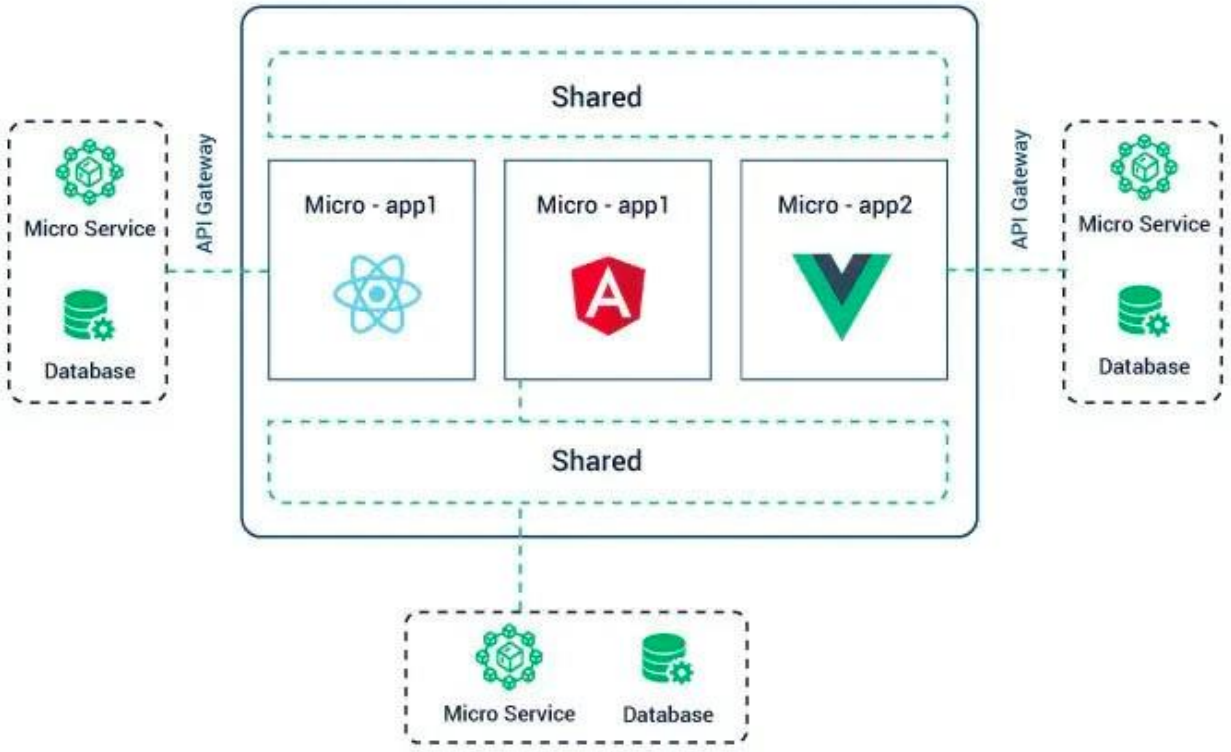

Рисунок — 2 Схема використання різних біблотек/фрейморків

На даній схемі показано, як різні бібліотеки та фреймворки можуть бути використані у мікрофронтендах.

Мікрофронтенди забезпечують ізольовану розробку, де кожен модуль може бути створений, протестований і розгорнутий незалежно. Це має ряд

переваг: Швидкість: Команди можуть швидше вносити зміни, оскільки вони працюють над меншими частинами системи. Надійність: Помилки в одному модулі не впливають на решту системи. Гнучкість: Легше адаптуватися до нових вимог або змін технологій.

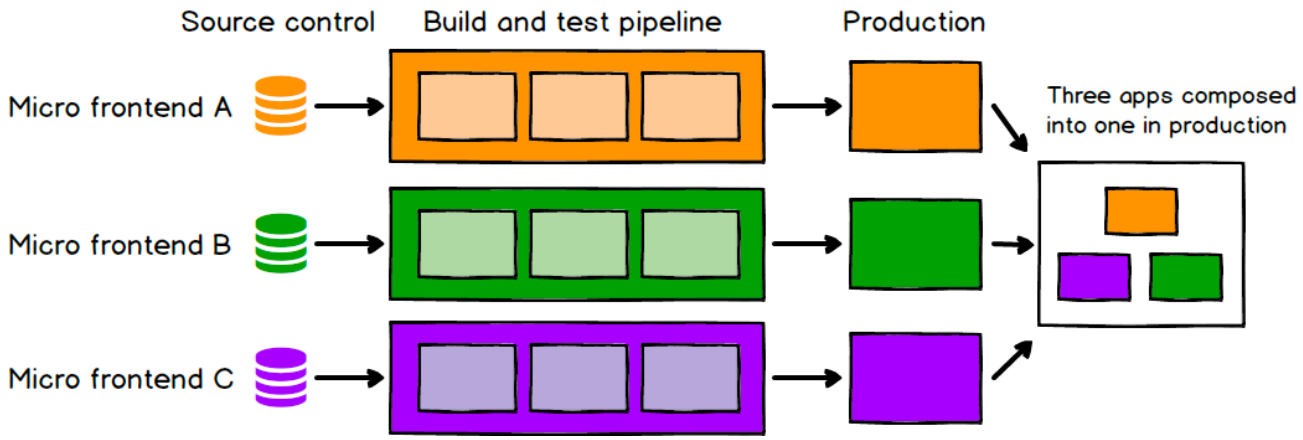

Рисунок — 3 Схема розгортання мікрофронтендів

Ця схема демонструє основні етапи розгортання мікрофронтендів та їх взаємодію.

Також сучасний ринок пропонує ряд інструментів та платформ для розробки на основі мікрофронтендів: Module Federation: Ця функція в Webpack дозволяє завантажувати код з різних джерел, що спрощує розподілену розробку.

Single-SPA: Це JavaScript-фреймворк, який дозволяє створювати навігаційні застосунки з різних мікрофронтендів. Tailor: Серверний компонент, який дозволяє з'єднувати різні мікрофронтенди в єдиний застосунок.

Таблиця 4

Порівняння інструментів для роботи з мікрофронтендами

Характеристика	Module Federation	Single-SPA	Tailor
Тип інструменту	Функція в Webpack	JavaScript-фреймворк	Серверний компонент
Основне	Завантаженн	Створення	З'єднан

призначення	я коду з різних джерел	навігаційних застосунків	ня мікрофронтендів
Основна перевага	Спрощення розподіленої розробки	Гнучкість у створенні навігації	Інтеграція різних мікрофронтендів
Типове використання	Великі проекти з розподіленими командами	Веб-застосунки з різноманітною навігацією	Великі застосунки з різними модулями

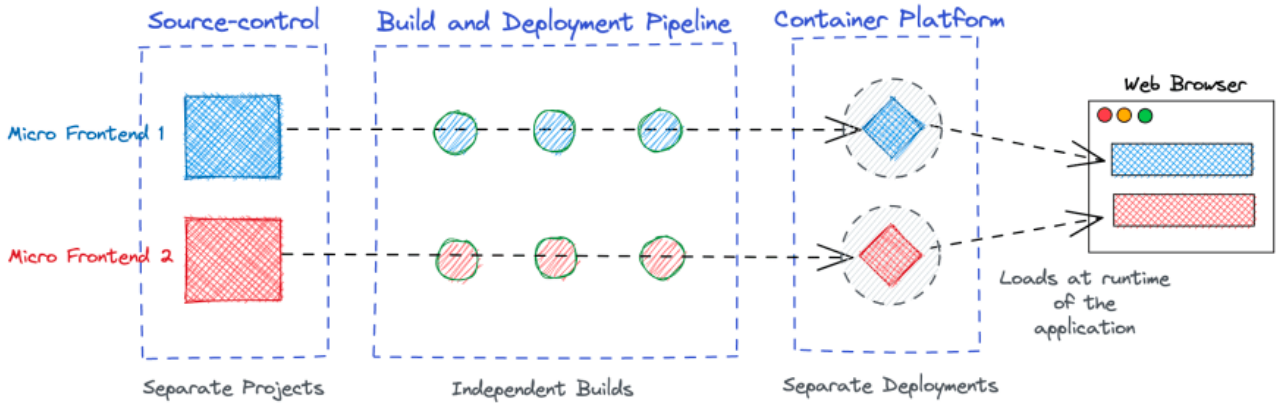

Рисунок — 4 Module federation

1.7 Сфери застосування мікрофронтендів

Мікрофронтенди, як модерна архітектурна парадигма, знаходять застосування в ряді різних сфер, де є потреба в гнучкості, масштабованості та автономності розробки. Розглянемо деякі основні сфери, де мікрофронтенди можуть принести найбільшу користь:

E-commerce платформи:

Сучасні електронні торгові платформи є складними системами, які включають велику кількість функціональних модулів: пошук товарів, рекомендаційні системи, корзина покупок, системи оплати, модулі відгуків і так далі. Мікрофронтенди дозволяють кожному з цих модулів розвиватися окремо. Наприклад, команда, що відповідає за корзину покупок, може вносити зміни і оптимізації без необхідності залучення інших команд, що пришвидшує інновації і впровадження нових функцій.

1. **Банківські та фінансові додатки:** Банківський сектор відомий своєю консервативністю, але в сучасних умовах він також потребує швидкої адаптації до змін. Нові банківські продукти, вимоги безпеки, персоналізований користувацький досвід — все це вимагає гнучкої архітектури. Мікрофронтенди дозволяють розбивати банківські додатки на модулі, такі як системи переказу коштів, кредитні модулі або інтерфейси управління рахунками.

2. **SaaS-платформи:**Множина функцій, які часто присутні в SaaS-додатках, таких як управління даними користувачів, інтеграція з іншими сервісами, налаштування і персоналізація, може бути виконана за допомогою мікрофронтендів. Це не тільки полегшує розробку, але і дозволяє клієнтам адаптувати додаток до своїх специфічних потреб.

3. **Платформи соціальних мереж:** Соціальні мережі є динамічними платформами з великою кількістю взаємодій користувачів. Від стрічки новин до системи сповіщень, від чатів до систем рекомендацій — мікрофронтенди можуть допомогти у виконанні різних задач, оптимізуючи взаємодію та забезпечуючи стабільність.

4. **Платформи для співпраці:** Додатки для командної роботи, такі як системи управління проектами, документообігу або віртуальних конференцій, мають ряд компонентів, що мають забезпечувати надійну роботу для великої кількості користувачів. Мікрофронтенди можуть

допомогти в оптимізації роботи цих компонентів, покращуючи продуктивність і користувацький досвід.

1.5. Аналіз програмного забезпечення для реалізації мікрофронтендів

Фреймворки та бібліотеки для розробки мікрофронтендів: React [3], створений Facebook, є одним з найбільш популярних JavaScript фреймворків для розробки користувацьких інтерфейсів. Завдяки своєму декларативному підходу і компонентній архітектурі, [7] React ідеально підходить для створення мікрофронтендів. Переваги: React [4] відомий своєю гнучкістю, високою продуктивністю та легкістю інтеграції з різними бекенд-технологіями. Завдяки віртуальному DOM, [7] React може швидко рендерити зміни без повного перезавантаження сторінки.

Недоліки: На відміну від більш повноцінних фреймворків, таких як Angular, React [4] є лише бібліотекою для розробки інтерфейсів, тому для реалізації більших додатків може знадобитися додаткові бібліотеки. Vue.js - це прогресивний JavaScript фреймворк, який також пропонує компонентний підхід до розробки.

Переваги: Vue.js славиться своєю легкістю та гнучкістю. Він має інтуїтивно зрозумілий синтаксис, а також надає розробникам гнучкість у виборі архітектурних рішень. Недоліки: [9] Vue.js може бути не таким зрілим або повноцінним, як деякі інші фреймворки, хоча й продовжує активно розвиватися.

Angular [15], розроблений Google, є повноцінним фреймворком для розробки веб-додатків.Переваги: Завдяки своєй модульній архітектурі та інтеграції з TypeScript, Angular надає розробникам потужні інструменти для створення масштабованих додатків.Недоліки: [9] Angular може мати вищий поріг входу для новачків через свою складність. Інструменти для інтеграції мікрофронтендів:

Module Federation: [7] Ця технологія дозволяє розділеним командам розробників завантажувати код з різних джерел, що полегшує інтеграцію

мікрофронтендів. Переваги: Module Federation дозволяє створювати гнучкі та масштабовані системи без необхідності перезавантаження всього додатку при змінах в одному з мікрофронтендів. Single SPA: Це маршрутизатор на рівні JavaScript, що дозволяє об'єднувати додатки, створені на різних фреймворках.Переваги: Single SPA спрощує розробку та випуск мікрофронтендів, надаючи єдиний пункт входу для всіх додатків. Tailor.js: Це рішення для сшивання мікрофронтендів на стороні сервера. Переваги: Завдяки Tailor.js, можливо динамічно об'єднувати різні частини додатку без необхідності змінювати основний код.

Платформи та сервіси для розгортання: Zeit Now: Ця платформа надає можливість автоматичного розгортання мікросервісів і мікрофронтендів.

Переваги: Zeit Now простий у використанні, інтегрується безпосередньо з GitHub, і надає автоматичне масштабування.

AWS Amplify: Сервіс від Amazon, що пропонує набір інструментів для розгортання та управління мікрофронтендами. Переваги: AWS Amplify має потужні засоби для масштабування, автоматичного розгортання та інтеграції з іншими службами AWS. Вивчення різних інструментів та технологій, доступних для розробки та розгортання мікрофронтендів, є ключовим для вибору оптимального стеку технологій для вашого проекту. Кожний інструмент має свої переваги та недоліки, тому важливо обрати той, який найкраще відповідає потребам вашої команди та проекту.

1.8 Використання мікрофронтендів разом з Next.js та Nx

[12] Next.js - це JavaScript фреймворк, створений на базі React, який надає можливості для створення статичних, серверно-рендерованих та інтерактивних веб-додатків.

Особливості: З автоматичним маршрутизатором, оптимізацією зображень, інтернаціоналізацією та API-маршрутами, Next.js пропонує комплексний набір

інструментів для сучасних веб-розробників. [13] Nx - це засіб розробки для монорепозиторіїв, який дозволяє організовувати, масштабувати і оптимізувати проекти на базі різних фреймворків та бібліотек. Особливості: Зі вбудованим кешуванням, залежностями між проектами та інтеграцією із засобами збірки, Nx полегшує управління великими проектами і сприяє повторному використанню коду

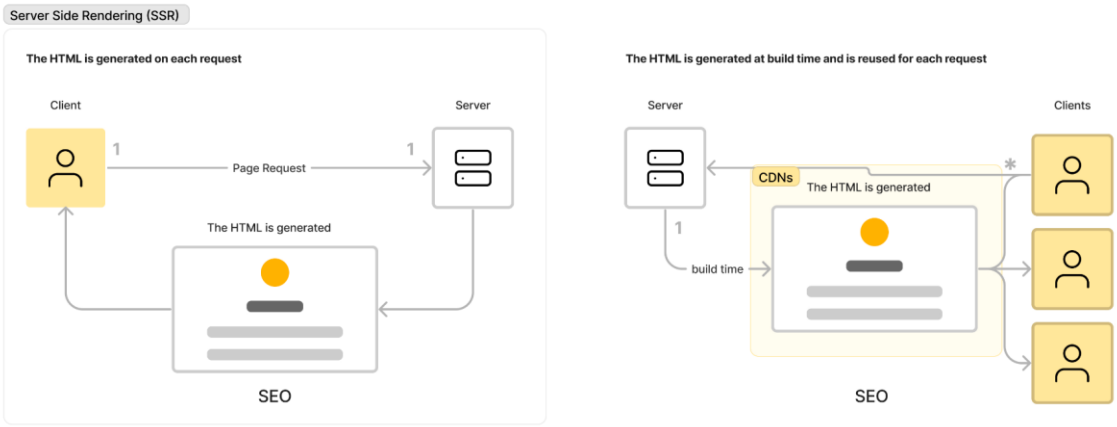

Рисунок — 5 Схема Next.js

2. Інтеграція мікрофронтендів з Next.js та Nx

Створення мікрофронтендів з Next.js:

[16] Next.js дозволяє ефективно розділяти логіку та відображення в мікрофронтендному підході, завдяки своїм вбудованим інструментам.

Детальний процес: Використання сторінок Next.js для створення окремих мікрофронтендів. Використання динамічного імпорту для лінивого завантаження компонентів. Використання API-маршрутів для обробки даних на сервері.

Організація мікрофронтендів з Nx: [16] Nx пропонує структурований підхід до управління мікрофронтендами в рамках єдиного монорепозиторію.

Детальний процес: Визначення залежностей між різними мікрофронтендами в Nx. Автоматизація процесів збірки та розгортання за допомогою засобів Nx.

Використання Nx Console для візуалізації та аналізу структури проекту.

3. Переваги комбінації Next.js та Nx для мікрофронтендів Масштабованість: Об'єднанням сил Next.js та Nx можна створити великі та масштабовані проекти, які легко підтримуються та розширюються. Повторне використання коду: Завдяки структурі монорепозиторіїв у Nx, команди можуть легко повторно використовувати код між різними мікрофронтендами, що забезпечує консистентність і ефективність розробки. Ефективна командна робота: Nx надає інструменти для спільної роботи команди, такі як паралельна збірка, ізольоване тестування та візуалізація залежностей, що полегшує спільну роботу над великими проектами.

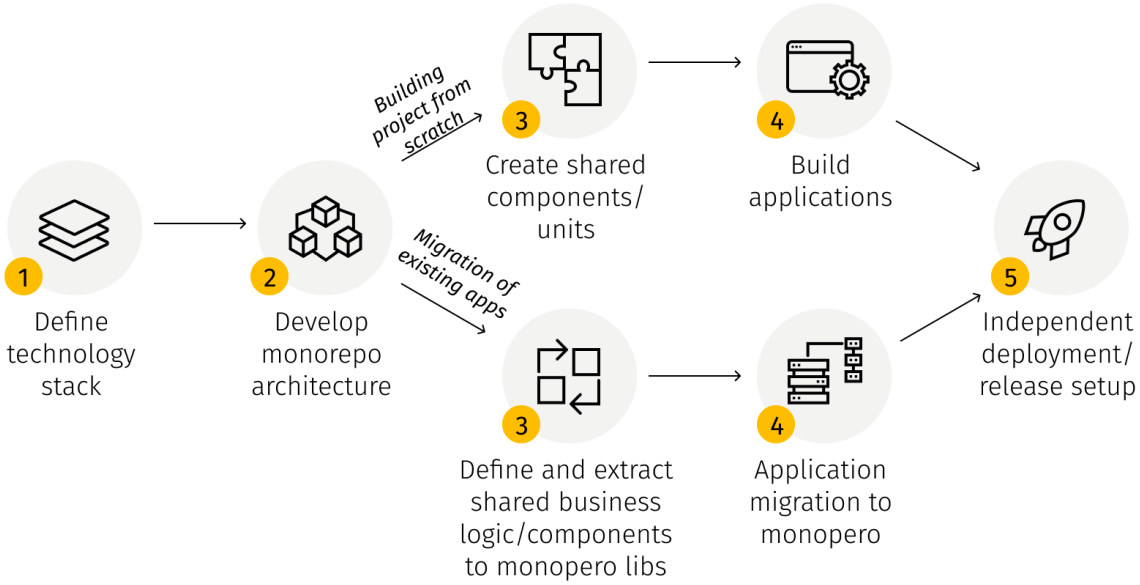

Рисунок — 6 Схема Nx monorepo

Використання мікрофронтендів разом з Next.js та Nx дозволяє командам розробників отримати максимальну вигоду від обох інструментів, забезпечуючи ефективність, швидкість та якість розробки.

1.9 Оцінка та аналіз ефективності побудування мікрофронтендів та їх вплив на продуктивність веб-застосунків

Побудова мікрофронтендів та їх вплив на продуктивність веб-застосунків є важливою темою в розробці програмного забезпечення. Давайте розглянемо більше деталей та переваги цього підходу:

Оцінка ефективності побудування мікрофронтендів: Швидкість розробки: Мікрофронтенди дозволяють розробничим командам працювати над окремими функціональними частинами веб-застосунку незалежно одна від одної. Це полегшує розробку, тестування та впровадження нового функціоналу. Внаслідок цього, цикли розробки стають коротшими, і продукт може швидше виходити на ринок. Масштабування: Мікрофронтенди дозволяють горизонтально масштабувати окремі частини системи, що спрощує рост системи з ростом навантаження. Якщо певна частина веб-застосунку отримує більше трафіку, можна просто масштабувати цей мікрофронтенд, не затрагуючи інші частини системи. Технічний борг: Завдяки ізольованому характеру мікрофронтендів, технічний борг може бути краще контрольований. Якщо потрібно оновити або замінити певний фрагмент фронтенду, це можна зробити без значного впливу на решту системи. Це сприяє підтримці високої якості коду та швидкому розвитку.

Вплив на продуктивність веб-застосунків: Завантаження: Мікрофронтенди можуть позитивно вплинути на час завантаження веб-застосунку, оскільки клієнти завантажують лише ті частини, які їм потрібні в даному контексті. Це зменшує обсяг завантажуваних ресурсів і поліпшує швидкість веб-сторінок. Взаємодія: Мікрофронтенди можуть працювати незалежно один від одного, що дозволяє користувачам взаємодіяти з окремими частинами веб-застосунку без

зависань або затримок. Це полегшує роботу з користувачем та покращує враження від використання додатк Оновлення: Можливість окремого оновлення мікрофронтендів сприяє постійному вдосконаленню додатку. Якщо потрібно внести зміни або виправлення, це можна зробити швидко та без впливу на інші частини системи. Це сприяє підтримці актуальності та стабільності додатку.

Загальний аналіз: Зазначена ефективність мікрофронтендів базується на досвіді багатьох компаній і великих проектів. Вони надають значні переваги в плані швидкості розробки, масштабування та управління технічним боргом. Однак важливо відзначити, що впровадження мікрофронтендів вимагає глибокого розуміння архітектури та правильного підходу. Необхідно добре продумати розподіл функціональності між мікрофронтендами та забезпечити їх взаємодію та сумісність для досягнення високої продуктивності веб-застосунків.

1.10 Методи та аналіз комунікації мікрофронтендних архітектур

В мікрофронтендних архітектурах (Micro Frontends), де функціональність веб-додатків розділена на окремі фрагменти, важливою є ефективна система комунікації між цими фрагментами. У цьому розділі ми розглянемо різні технології та методи комунікації в мікрофронтендних архітектурах, включаючи їх переваги та недоліки.

Методи Комунікації: Одним із найпоширеніших методів комунікації в мікрофронтендних архітектурах є передача властивостей (props) від контейнерного фронтенду до вкладених мікрофронтендів. Цей метод базується на концепції компонентної архітектури та дозволяє передавати дані та функції між фрагментами додатку.

Переваги:

- Простий у реалізації.
- Використовується у багатьох фреймворках, зокрема у React.

- Можливість уникнути проблем "prop drilling" за допомогою React Context та інших рішень.

Недоліки:

- Збільшує зв'язаність між мікрофронтендами.
- Вимагає використання одного фреймворку для всіх фронтенд-частин.

Іншим способом комунікації є використання платформеного сховища, такого як Local Storage у веб-браузерах або Async Storage для мобільних додатків. Кожен мікрофронтенд може читати та записувати дані в це сховище, дозволяючи спільно використовувати дані між фрагментами.

Переваги:

- Підтримується як для веб-браузерів, так і для мобільних платформ.
- Зменшує зв'язаність між фронтенд-частинами.

Недоліки:

- Не підходить для обміну великими об'ємами даних.
- Може бути складним у використанні, коли дані змінюються динамічно.

Ще одним методом комунікації є використання подій та системи подій браузера. Кожен мікрофронтенд може створювати та підписуватися на події, що дозволяє спільно використовувати дані та взаємодіяти між фрагментами.

Переваги:

- Реалізовано за допомогою нативних засобів браузера.
- Легко розширюється та масштабується.

Недоліки:

- Не підходить для мобільних додатків.
- Потребує додаткового коду для управління підпискою на події та їх відпискою.

Аналіз Типів Комунікації

Пряма взаємодія полягає у взаємодії між мікрофронтендами без посередника. Вона може бути реалізована за допомогою методів, наведених вище, і дозволяє фрагментам взаємодіяти напряму. Цей підхід забезпечує велику гнучкість, але може призвести до складностей у відслідковуванні та керуванні взаємозв'язками між фрагментами. Посередник контейнеру — це фрагмент, який служить посередником між іншими мікрофронтендами. Він може приймати дані від одного фрагмента та передавати їх іншому, використовуючи один із методів комунікації. Цей підхід дозволяє зменшити зв'язаність між фрагментами та спростити управління взаємодією. Використання Зовнішніх Інструментів:

Module Federation - це концепція та плагін для бандлера, який дозволяє фрагментам коду динамічно завантажуватися та взаємодіяти між собою. Цей метод полегшує створення високомасштабованих мікрофронтендів.

Переваги:

- Зменшує зв'язаність між мікрофронтендами.
- Дозволяє динамічно завантажувати код фрагментів.

Недоліки:

- Вимагає підтримки конкретного бандлера (наприклад, Webpack).
- Може потребувати додаткового налаштування.

Apache Kafka - це розподілена платформа для обміну повідомленнями, яка дозволяє мікрофронтендам обмінюватися даними та подіями в реальному часі.

Переваги:

- Висока масштабованість та надійність.
- Забезпечує обмін даними в реальному часі.

Недоліки:

- Вимагає складного налаштування та адміністрування.

- Може бути зайвим для менших проектів або тих, які не потребують обміну даними в реальному часі.

3. Event Bus:

Event Bus - це механізм для реалізації асинхронної комунікації між мікрофронтендами за допомогою подій. Кожен фрагмент може публікувати події в шину подій, а інші фрагменти можуть підписуватися на ці події та реагувати на них. Це дає можливість для локального та глобального спілкування між фрагментами.

Переваги:

- Забезпечує асинхронну та подійну комунікацію.
- Дозволяє фрагментам взаємодіяти без прямої залежності один від одного.

Недоліки:

- Потребує встановлення та налаштування системи подій.
- Потребує управління підписками на події та їх відписками для попередження витоку пам'яті.

Вибір конкретного методу повинен базуватися на вимогах вашого проекту та ваших можливостях для налаштування та розробки. Важливо враховувати переваги та недоліки кожного методу для досягнення ефективної системи комунікації між мікрофронтендами.

1.11 Висновки до розділу 1

1. В сучасному світі веб-розробки стрімкий розвиток технологій призводить до потреби в гнучких рішеннях для складних архітектур, як-от мікрофронтенди, що дозволяють оптимізувати роботу команди та прискорюють внесення змін.
2. Головний виклик у впровадженні мікрофронтендів полягає в забезпеченні ефективної комунікації між різними частинами системи, не

втрачаючи при цьому ізольованість та незалежність кожного з компонентів.

3. Розглянуті методи комунікації між мікрофронтендами вказують на різноманітність підходів, починаючи від глобального об'єкта стану і закінчуючи спеціалізованими рішеннями на основі подій.

4. Незважаючи на існуючі бібліотеки та фреймворки, що спрощують впровадження мікрофронтендів, важливо розуміти основні принципи їх роботи та враховувати специфіку конкретного проекту.

5. Вибір методу комунікації між мікрофронтендами має базуватися на аналізі потреб проекту, можливостях команди та перевагах та недоліках кожного з розглянутих підходів.

РОЗДІЛ 2 ТЕХНІЧНІ АСПЕКТИ ТА ПРАКТИЧНЕ ЗАСТОСУВАННЯ МІКРОФРОНТЕНДНИХ АРХІТЕКТУР

2.1 Технології та інструменти для розробки мікрофронтендів

[2]Мікрофронтенди стали популярним підходом до розробки веб-застосунків, оскільки вони дозволяють командам розробляти, тестувати та розгортати частини фронтенду незалежно одна від одної. Цей підхід пропонує гнучкість та ефективність, особливо для великих проектів з численними командами розробників. На сьогоднішній день існує безліч технологій для реалізації мікрофронтендів, проте серед них React, Vue.js та Angular вважаються найбільш популярними.

[10]React, який було розроблено компанією Facebook, використовує компонентний підхід, що дозволяє розробникам створювати повторювані елементи користувацького інтерфейсу. З іншого боку, Vue.js пропонує реактивний підхід до управління даними, що робить його особливо зручним для створення динамічних веб-інтерфейсів. Angular, платформа розроблена Google, відзначається своєю масштабованістю та надає повний набір інструментів для розробки веб-застосунків, від маршрутизації до управління станом.

Під час створення мікрофронтендного застосунку важливо ретельно підготувати проект. Це передбачає створення нового каталогу для застосунку та налаштування нового проекту на React, будь то за допомогою Create React App, Next.js чи будь-якого іншого інструменту. Після цього потрібно визначити логічні розділення в межах застосунку, де кожне таке розділення представлятиме окремий мікрофронтенд.

Однією з ключових особливостей мікрофронтендів є їх здатність до комунікації між собою. Зазвичай це досягається за допомогою користувацьких подій, бібліотек управління станом або інших шаблонів комунікації, таких як Pub/Sub.

Контейнерний застосунок відіграє важливу роль, виступаючи як оркестратор для різних мікрофронтендів. Він має містити маршрутизацію, щоб на основі URL завантажувати відповідний мікрофронтенд. При цьому мікрофронтенди повинні динамічно завантажуватися в контейнерний застосунок.

Також необхідно звертати увагу на залежності. Спільні залежності між мікрофронтендами повинні бути ідентифіковані та розділені, щоб зменшити розмір пакета та забезпечити узгодженість між застосунками. Що стосується стилів, то важливо створити спільну бібліотеку стилів або забезпечити, щоб кожен мікрофронтенд інкапсулював свої стилі, щоб уникнути конфліктів.

У контексті тестування, важливо забезпечити, що кожен мікрофронтенд працює належним чином як самостійно, так і при інтеграції в контейнерний застосунок. І, нарешті, потрібно налаштувати CI/CD конвеєр для автоматичної збірки, тестування та розгортання кожного мікрофронтенда.

2.1.1 Використання Nx monorepo для організації мікрофронтендних архітектур на React.

В усіх сферах програмування масштабованість та ефективність розробки залишаються в центрі уваги, а мікрофронтенди стали одним з популярних відповідей на ці виклики. В контексті розробки на React [18], інструмент Nx пропонує ефективний підхід до управління масштабованими проектами завдяки своїй monorepo структурі.Мікрофронтенди, як архітектурний підхід, набирають популярності завдяки своїй спроможності розбивати фронтенд застосунки на менші, незалежні частини. Ця модульність дозволяє командам розробляти,

тестувати та розгортати частини застосунку автономно. Однак така структура може призвести до викликів у управлінні кодом, залежностями та іншими ресурсами. Тут вступає Nx. Що таке Nx? Nx — це інструмент для створення monorepo робочих просторів. Він був розроблений командою Narwhal Technologies, спочатку як інструмент для Angular. Але завдяки його гнучкості та модульності, він швидко адаптувався для підтримки інших фреймворків, зокрема React.

Переваги використання Nx для мікрофронтендів на React:

1. Уніфікація коду: Великий плюс Nx полягає в тому, що весь код зберігається в одному репозиторії. Це не тільки спрощує управління кодом, але також полегшує спільну роботу різних команд над спільними бібліотеками.
2. Оптимізоване тестування: Завдяки стратегії "affected" в Nx, тести та збірки виконуються лише для тих частин застосунку, які зазнали змін. Це може значно прискорити цикли розробки, особливо в великих проектах.
3. Інтеграція з іншими інструментами: Nx працює ефективно з різноманітними інструментами, такими як Jest для юніт-тестування та Cypress для енд-ту-енд тестування. Така інтеграція забезпечує гладкий робочий процес для розробників.

Недоліки:

1. Крива навчання: Для команд, незнайомих з monorepo або Nx, може знадобитися час для навчання. Хоча документація Nx є досить зрозумілою, існують певні особливості, які потребують додаткового вивчення.
2. Потенційні проблеми з продуктивністю: Якщо не використовувати оптимізовані стратегії, такі як "affected", великі monorepo можуть збільшити час збірки та розгортання.

Аналіз інших аналогів: Хоча існують інші інструменти для управління monorepo, такі як Lerna та Yarn Workspaces, Nx виділяється завдяки своєму фокусу на продуктивність та інтеграцію з різноманітними фреймворками. Чому варто використовувати Nx для мікрофронтендів на React? Nx не тільки допомагає управляти кодом та ресурсами в масштабованих проектах, але й пропонує набір інструментів, які оптимізують робочий процес розробки. Для команд, що прагнуть до ефективності, гнучкості та масштабованості, Nx може бути відмінним вибором для їхніх проектів на React

2.2 Рендерінг мікрофронтендів у браузері

У сучасному світі веб-розробки рендерінг стає не лише про відображення контенту, але й про інтеграцію різних частин системи в єдину цілісність. Рендерінг мікрофронтендів втілює цю ідею, поєднуючи розподілені частини веб-застосунку в єдиний користувацький інтерфейс. Цей процес є набагато складнішим, ніж традиційний рендерінг, через розподілену природу мікрофронтендів.

Мікрофронтенди розробляються таким чином, що кожна їх частина може існувати автономно, мати свої залежності, стан та життєвий цикл. Ця автономія дозволяє розробникам вибирати технології, які найкраще підходять для конкретних завдань, незалежно від решти системи. Однак ця відокремленість також призводить до викликів, пов'язаних з координацією та інтеграцією цих автономних частин у єдиний застосунок. Коли мова йде про рендерінг мікрофронтендів у браузері, основна увага приділяється оптимізації завантаження, взаємодії з глобальним середовищем браузера та управлінню залежностями між різними частинами системи. Кожен браузер має свої особливості, які можуть впливати на процес рендерингу. Додатково, сучасні браузери, такі як Chrome, Firefox, Edge або Safari, надають розробникам інструменти для глибокого аналізу рендерингу. Це може включати в себе аналіз

виконання JavaScript, оптимізацію завантаження ресурсів та інші аспекти, які впливають на продуктивність та користувацький досвід. Рендерінг мікрофронтендів у браузері також має декілька особливостей, пов'язаних з потребою забезпечити ізоляцію між різними частинами застосунку, а також з гарантуванням цілісності та стабільності системи в цілому. В узагальненому вигляді, рендерінг мікрофронтендів є процесом, який вимагає глибокого розуміння як архітектурних особливостей, так і конкретних властивостей браузера. Це поєднання технічних та архітектурних аспектів робить рендерінг мікрофронтендів унікальним викликом для розробників.

2.2.1 Особливості рендерингу мікрофронтендів

У контексті мікрофронтендів, рендерінг може відбуватися на декілька способів:

Клієнтський рендерінг (CSR): Весь контент завантажується динамічно на стороні клієнта. Це дозволяє додавати або видаляти мікрофронтенди без перезавантаження сторінки. Проте, цей метод може бути менш ефективним для SEO та початкової завантажуваності. Серверний рендерінг (SSR): Контент генерується на сервері та відправляється на клієнта у вигляді готового HTML. Це покращує початкову завантажуваність та SEO, але може ускладнити динамічні взаємодії на сторінці. Статичний рендерінг (SSG): Весь контент генерується під час збірки застосунку і служиться як статичні файли. Це найшвидший спосіб служіння контенту, але він менш гнучкий щодо динамічних взаємодій.

Окрім методів рендерингу, існує декілька ключових аспектів, які впливають на рендерінг мікрофронтендів: Оптимізація завантаження: щоб забезпечити швидке завантаження мікрофронтендів, можна використовувати техніки, такі як дерево-лускання (tree-shaking), код-спліттинг, асинхронне завантаження модулів тощо. Кешування: Для покращення продуктивності можна кешувати частини мікрофронтендів, що не змінюються часто. Взаємодія між

мікрофронтендами: Коли декілька мікрофронтендів працює разом на одній сторінці, вони повинні взаємодіяти між собою, щоб надавати злагоджений користувацький досвід. Окрім того, існує ряд викликів, пов'язаних з рендерингом мікрофронтендів: Сумісність стилів: Оскільки кожен мікрофронтенд може мати свої стилі та бібліотеки, вони можуть конфліктувати між собою на сторінці. Спільний стан: Якщо декілька мікрофронтендів повинні спільно використовувати який-небудь стан, це може призвести до складностей в управлінні цим станом. Залежності: Мікрофронтенди можуть мати спільні залежності, які повинні бути оптимізовані, щоб уникнути дублювання. Для реалізації ефективного рендерингу мікрофронтендів необхідно ретельно планувати архітектуру, вибирати відповідні технології та інструменти та постійно моніторити продуктивність застосунку. золяція мікрофронтендів: Ізоляція є ключовою особливістю мікрофронтендів. Вона забезпечує, що кожен мікрофронтенд може працювати незалежно від інших, з своїм власним контекстом, залежностями і ресурсами. Це не тільки спрощує розробку та тестування, але також забезпечує гнучкість в розгортанні та масштабуванні.

Композиція мікрофронтендів: На відміну від традиційних односторінкових застосунків, де весь контент завантажується разом, мікрофронтенди дозволяють динамічно компонувати контент на сторінці, базуючись на бізнес-логіці, конфігурації або користувацьких налаштувань. Процес рендерингу: Якщо дивитися з технічної точки зору, процес рендерингу мікрофронтенду може бути подібний до рендерингу звичайного веб-компоненту. Однак, особливість полягає в тому, як ці мікрофронтенди взаємодіють між собою та з головним застосунком. Динамічне завантаження та рендерінг: Однією з ключових особливостей мікрофронтендів є можливість динамічного завантаження та рендерингу. Замість завантаження всіх ресурсів застосунку під час початкового завантаження, мікрофронтенди можуть бути завантажені та відображені за потребою.

Оптимізація завантаження: Оптимізуючи завантаження мікрофронтендів, можна забезпечити швидше завантаження застосунку. Це може бути досягнуто за допомогою технік, таких як ліниве завантаження, асинхронне завантаження та кодовий розподіл.

Управління станом: Хоча кожен мікрофронтенд може мати свій власний локальний стан, може виникнути потреба у спільному стані між різними мікрофронтендами. Це може потребувати використання глобальних систем управління станом або спеціальних шаблонів для спілкування між мікрофронтендами.

Безпека та ізоляція: Оскільки мікрофронтенди можуть розроблятися різними командами або навіть організаціями, виникає потреба в ізоляції та безпеці. Це забезпечує, що мікрофронтенди не можуть неправильно взаємодіяти або впливати на інші частини системи. Продовжуючи розглядати рендерінг мікрофронтендів, важливо звернути увагу на ряд ключових проблем та рішень, що виникають в процесі розробки та впровадження цих систем. Ці аспекти включають в себе оптимізацію продуктивності, управління залежностями, комунікацію між мікрофронтендами та багато іншого.Глобальне середовище: Однією з найбільших проблем при розробці мікрофронтендів є глобальне середовище браузера. Оскільки браузер ділиться між всіма мікрофронтендами, існує ризик конфліктів, особливо щодо глобальних змінних, стилів та подій. Це вимагає від команд розробників бути обережними при взаємодії з глобальними ресурсами.

Модульна архітектура: Для оптимального рендерингу мікрофронтендів необхідно розробляти модульну архітектуру. Це дозволяє застосунку завантажувати тільки ті частини, які потрібні для конкретної сторінки або функції, забезпечуючи швидше завантаження та кращу продуктивність. правління ресурсами: З огляду на те, що мікрофронтенди можуть розроблятися різними командами з використанням різних бібліотек і фреймворків, виникає

потреба в ефективному управлінні ресурсами. Це включає в себе оптимізацію завантаження, кешування та інші техніки для забезпечення гладкої роботи застосунку.

Респонсивний дизайн: У світі мобільних пристроїв та різноманітних розмірів екранів, респонсивний дизайн стає ключовим. Мікрофронтенди повинні бути гнучкими і адаптивними до різних типів пристроїв і розмірів екранів. Поведінка на клієнті та сервері: Залежно від вимог до продуктивності та доступності, мікрофронтенди можуть використовувати комбіновані підходи до рендерингу на стороні клієнта та сервера, забезпечуючи оптимальний баланс між швидкістю завантаження, динамічною взаємодією та SEO. Тестування та надійність:

Оскільки мікрофронтенди розробляються незалежно, вони також повинні бути протестовані незалежно. Однак, коли вони взаємодіють у реальному застосунку, можуть виникнути проблеми інтеграції. Це вимагає комплексного тестування та моніторингу для забезпечення надійності рендерингу в умовах реального світу.

Масштабування та оптимізація: З ростом складності застосунку та кількості користувачів може виникнути потреба в оптимізації рендерингу. Це може включати в себе техніки, такі як віртуалізація DOM, оптимізація взаємодії з сервером, а також використання CDN для швидкого доступу до ресурсів.

Продовжуючи дослідження рендерингу мікрофронтендів, необхідно звертати увагу на те, як різні техніки та підходи можуть бути адаптовані для конкретних потреб бізнесу та користувачів. При правильному підході, мікрофронтенди можуть забезпечити високу продуктивність, гнучкість та стабільність для великих веб-застосунків.

2.2.2 Виклики та рішення під час рендерингу.

Рендерінг в контексті мікрофронтендної архітектури може виявитися складнішим завданням, ніж у традиційних монолітних застосунках. Особливості

рендерингу мікрофронтендів, які були виділені у дослідницькому матеріалі, вказують на конкретні виклики та відповідні рішення. Одним з ключових викликів є ізоляція різних мікрофронтендів від глобального середовища браузера. Браузери мають глобальне середовище, що може призводити до конфліктів, особливо при використанні ресурсів, таких як CSS та JavaScript. Така ізоляція важлива для забезпечення незалежності та стабільності кожного мікрофронтенду.Відповідно до дослідницького матеріалу, для вирішення цього виклику може бути застосовано стратегію "поділу коду", яка передбачає розбиття коду на менші частини, які можуть бути завантажені окремо. Це може допомогти уникнути завантаження непотрібного коду та забезпечити швидший час завантаження. Додатково, інтеграція між різними мікрофронтендами може стати викликом, зокрема коли мова йде про спільне використання даних чи стану. Однак, за допомогою спеціалізованих бібліотек та інструментів, таких як Module Federation у Webpack, можливо створити спільний контекст між мікрофронтендами. Ще одним важливим аспектом, який був підкреслений у дослідницькому матеріалі, є необхідність оптимізації рендерингу для різних браузерів. Сучасні браузери, такі як Chrome, Firefox, Edge або Safari, мають свої особливості та відмінності в рендерингу. Розробники повинні бути готові адаптувати свій код для забезпечення оптимальної продуктивності в кожному браузері. В узагальненому вигляді, рендерінг мікрофронтендів вимагає глибокого розуміння не тільки основних принципів веб-розробки, але і специфіки мікрофронтендного підходу. Успішна реалізація такої системи вимагає комбінації правильної архітектури, вибору технологій та неперервної адаптації до змінних умов середовища.Враховуючи складність рендерингу мікрофронтендів, важливо розглянути й інші аспекти, що можуть впливати на ефективність та якість цього процесу.

Адаптивний дизайн: Якщо розглядати сучасний веб, адаптивний дизайн вже давно не є просто "хорошою практикою", але обов'язковим аспектом. В

контексті мікрофронтендів це стає ще більш актуальним, оскільки кожен компонент може мати свої особливості відображення на різних пристроях. Тому необхідно забезпечити, щоб рендеринг відповідав потребам користувачів, незалежно від того, який пристрій вони використовують.

Динамічний контент: Однією з ключових особливостей сучасних веб-додатків є їхня динамічність. Мікрофронтенди часто використовуються для побудови додатків, які активно взаємодіють з користувачем у реальному часі. Отже, процес рендерингу повинен бути достатньо гнучким, щоб адаптуватися до постійно змінюваного контенту.

Взаємодія з сервером: Мікрофронтенди часто працюють у парі з мікросервісами на бекенді. Це означає, що процес рендерингу також повинен враховувати взаємодію з сервером, оптимізацію запитів, обробку даних та інші аспекти, пов'язані з серверною частиною.

Процес розробки: Рендерінг мікрофронтендів не обмежується тільки виведенням контенту на екран. Це також стосується процесу розробки, де розробники повинні мати можливість легко тестувати, налагоджувати та оптимізувати рендеринг. У цьому контексті важливо мати інструменти та практики, які допоможуть команді ефективно працювати над рендерингом. В узагальненому вигляді, рендерінг мікрофронтендів є надзвичайно складним завданням, яке вимагає комплексного підходу. Щоб досягти успіху в цьому напрямку, команди повинні бути готові вивчити всі аспекти цього процесу і адаптувати свої методи розробки до специфіки мікрофронтендної архітектури.Враховуючи множинність аспектів рендерингу мікрофронтендів, додаткові роздуми про технічні та архітектурні виклики можуть привести до глибшого розуміння цієї проблеми. Оптимізація завантаження: Оскільки мікрофронтенди можуть бути розроблені різними командами із застосуванням різних технологій, існує ризик завантаження великої кількості повторюваного або непотрібного коду. Інтелектуальне розбиття коду та динамічне завантаження

модулів може допомогти оптимізувати завантаження, забезпечуючи тільки необхідний код для конкретного користувацького сценарію. Забезпечення консистентності: Якщо мікрофронтенди розроблялися незалежно, можливі стилістичні та функціональні невідповідності. Це може призвести до неконсистентного користувацького досвіду. Використання спільних компонентів дизайну або наборів UI може допомогти забезпечити узгодженість візуального представлення та взаємодії.

Стабільність і відмовостійкість: При інтеграції багатьох мікрофронтендів у єдиний застосунок існує ризик, що неполадки в одному мікрофронтенді можуть вплинути на стабільність інших. Тому важливо розглядати стратегії обробки помилок та механізми відновлення. Взаємодія з третіми сторонами: Мікрофронтенди можуть взаємодіяти з різними зовнішніми службами або API. Це може призвести до додаткових викликів, пов'язаних із безпекою, крос-доменними запитами та іншими питаннями.

Масштабованість: З ростом застосунку та збільшенням кількості мікрофронтендів може з'явитися потреба в додаткових оптимізаціях для забезпечення гладкої роботи застосунку на різних пристроях і платформах.

Адаптація до нових технологій: Світ веб-технологій постійно змінюється. Мікрофронтендна архітектура повинна бути гнучкою, щоб адаптуватися до нових технологій, стандартів та підходів.

У висновку, рендерінг мікрофронтендів є викликом, який вимагає глибокого аналітичного підходу, дослідження та неперервних ітерацій. Але з правильною стратегією та підходом, можливо досягти високої продуктивності, стабільності та консистентності у великих веб-застосунках.

2.2.3 Взаємодія мікрофронтендів під час рендерингу.

У контексті мікрофронтендної архітектури взаємодія між різними частинами системи є відділеною, але водночас критично важливою. Всі

компоненти мають взаємодіяти гладко, але незалежно, щоб забезпечити оптимальний користувацький досвід. Спільний контекст: Однією з ключових особливостей мікрофронтендів є їх здатність працювати в спільному контексті. Це означає, що, хоча кожен мікрофронтенд може бути розроблений незалежно, вони повинні ділити певний набір даних та стану. Таке спільне середовище забезпечує консистентність досвіду для користувача та дозволяє різним частинам системи взаємодіяти ефективно.

Комунікація між мікрофронтендами: Для забезпечення надійної та ефективної взаємодії між мікрофронтендами важливо мати чітко визначені механізми комунікації. Це може бути реалізовано за допомогою подій, посланників, сервісних робітників або інших патернів взаємодії. Синхронізація стану: Оскільки кожен мікрофронтенд може мати свій власний локальний стан, виникає потреба в його синхронізації з іншими частинами системи. Тут можуть бути застосовані різні підходи, такі як централізоване управління станом або розподілені механізми подій. Управління залежностями: В процесі рендерингу мікрофронтенди можуть взаємодіяти з рядом зовнішніх бібліотек та залежностей. Забезпечення їхньої сумісності та консистентності є важливим завданням, що вимагає додаткової уваги під час розробки. Оптимізація процесу взаємодії: Щоб забезпечити плавність рендерингу та відгук системи на дії користувача, необхідно оптимізувати процес взаємодії між мікрофронтендами. Це може включати в себе декомпозицію завдань, асинхронну обробку або використання віртуальних DOM-структур для мінімізації дорогоцінних операцій в DOM. В узагальненому вигляді, взаємодія мікрофронтендів під час рендерингу є важливим аспектом архітектури. Це вимагає від розробників глибокого розуміння принципів взаємодії, а також здатності адаптувати ці принципи до специфіки мікрофронтендної архітектури.

2.3 Переваги та недоліки мікрофронтендних архітектур.

У великому і розгалуженому світі веб-розробки мікрофронтенди стали якорем, що допомагає компаніям ефективно масштабувати та адаптуватися до змінюваних вимог. Але як із будь-якою технологічною стратегією, вони мають свої світлі і темні сторони. При детальному розгляді автономності команд, можна побачити, що мікрофронтенди не просто розділяють веб-застосунок на частини.

Вони відкривають шлях до культури, де команди можуть діяти незалежно, вибираючи свої власні шляхи до досягнення цілей. Це створює екосистему інновацій, де кожна команда може ризикувати, експериментувати та вчитися, не піддаючи під загрозу загальний продукт. Технологічна гнучкість є ще однією ключовою особливістю. [6] У світі, де нові фреймворки та інструменти з'являються майже щодня, можливість вибирати та адаптувати новітні технології є неоціненною. Це не лише дозволяє командам залишатися на передовій технологічного прогресу, але і підтримує застосунок актуальним і конкурентоспроможним.

Масштабованість є основою більшості сучасних веб-застосунків. З ростом та розвитком додатку, масштабованість стає невід'ємною частиною його успіху. Мікрофронтенди пропонують структуру, де нові функції можуть бути легко інтегровані без переробки основної архітектури. Однак, незалежність, яку пропонують мікрофронтенди, може призвести до складності управління. Це не завжди легко забезпечити, щоб усі частини гармонійно працювали разом, особливо коли вони розроблені різними командами з різними технологічними стеками. Потенційне збільшення навантаження на клієнта також є важливим аспектом.

Поглибимо наш розуміння переваг та недоліків мікрофронтендних архітектур, розглядаючи конкретні приклади, аналізуючи їх та порівнюючи з традиційними підходами. Автономність команд: Уявімо стартап, що розробляє соціальний медіа додаток. Завдяки мікрофронтендам команда, відповідальна за

стрічку новин, може вносити зміни та вдосконалення незалежно від команди, що працює над месенджером. Це призводить до того, що обидві команди можуть швидше реагувати на відгуки користувачів та вносити покращення. У традиційній монолітній архітектурі така незалежність була б майже неможливою, оскільки зміни в одній частині системи могли б впливати на інші.

Технологічна гнучкість: Коли новий фреймворк або інструмент з'являється на ринку, команди, що працюють з мікрофронтендами, можуть швидко його інтегрувати в свою частину продукту, не турбуючись про інші частини. Наприклад, якщо команда вирішила перейти з React на Vue для покращення продуктивності своєї частини застосунку, вона може це зробити без великих труднощів. У монолітному застосунку такий перехід потребував би координації всієї команди та значних зусиль для переписування коду.

Масштабованість: У великих організаціях, де додаток постійно розширюється, мікрофронтенди дозволяють легко додавати нові модулі. Наприклад, електронний магазин може легко додати новий модуль для віртуальної примірки одягу, без перебудови всього застосунку. Тоді як в монолітній структурі, інтеграція такого великого нового модулю могла б стати викликом і потребувати значних змін в основному коді.

Складність управління: Хоча мікрофронтенди пропонують багато переваг, управління такою розподіленою системою може бути викликом. Для прикладу, у великому фінансовому застосунку, де безпека є ключовою, координація між мікрофронтендами для забезпечення цілісності та безпеки даних може бути важкою.

Потенційне збільшення навантаження на клієнта: Якщо кожен мікрофронтенд завантажує свої власні бібліотеки та ресурси, це може призвести до збільшення часу завантаження сторінки. Наприклад, якщо новинний сайт має окремі мікрофронтенди для відображення новин, коментарів та реклами, і кожен

з них завантажує свої власні стилі та скрипти, це може знизити продуктивність та швидкість відгуку сайту.

Виклики інтеграції: Інтеграція різних мікрофронтендів у єдиний додаток може призвести до неконсистентності в дизайні та користувацькому досвіді. Уявімо онлайнову платформу навчання, де один мікрофронтенд відповідає за відео-уроки, а інший за текстовий контент. Якщо ці два мікрофронтенди були розроблені різними командами з різними візуальними стандартами, студенти можуть відчувати неконсистентність під час переходу від одного розділу до іншого.

Враховуючи ці приклади, ми можемо бачити, що мікрофронтенди пропонують унікальний набір можливостей і викликів. Важливо розуміти, як максимізувати їхні переваги та мінімізувати недоліки для досягнення найкращих результатів у конкретному проекті.

Важливо гармонійно інтегрувати мікрофронтенди, щоб не перевантажити клієнтську сторону завеликою кількістю запитів або завеликим об'ємом завантажуваного коду. Наостанок, виклики інтеграції можуть виявитися досить складними. Кожен мікрофронтенд, будучи незалежною сутністю, повинен бути правильно вбудований в загальний додаток, що вимагає додаткових зусиль для забезпечення єдності користувацького досвіду.

2.3.1 Виклики при інтеграції та розгортанні мікрофронтендів.

Мікрофронтенди, безумовно, пропонують ряд переваг, особливо для великих застосунків та команд. Однак як і з будь-якою новою технологією або архітектурним підходом, є певні виклики, які потрібно враховувати, особливо коли справа доходить до інтеграції та розгортання.

Інтеграція різних технологічних стеків може виявитися досить викликовою. На основі прикладу з дипломної роботи, якщо одна команда використовує React для своєї частини застосунку, а інша вибирає Angular, можуть

виникнути проблеми зі сумісністю, зокрема в контексті спільних бібліотек або залежностей.

Координація між командами стає критичною, особливо в контексті неперервної інтеграції та неперервного розгортання (CI/CD). Коли кожна команда може вносити зміни в свій мікрофронтенд незалежно, важливо мати процеси та інструменти, які гарантують, що ці зміни не порушують загальну функціональність застосунку. Основуючись на принципах CI/CD, мікрофронтенди пропонують революційний підхід до розробки, зосереджуючись на модульності та автономності. Ця автономність дає можливість командам розробників працювати паралельно над різними частинами застосунку, що забезпечує більш швидке та ефективне впровадження нововведень.

Водночас, автоматизація, яка лежить в основі CI/CD, гарантує, що код, який інтегрується в основний продукт, пройшов всі необхідні етапи тестування. Це зменшує ризик виникнення дефектів у продакшені. Так, наприклад, при розробці нового мікрофронтенду, інструменти, такі як Travis CI або CircleCI, можуть автоматично запускати набори тестів для перевірки якості коду та його сумісності з існуючими системами. Проте, хоча автоматизація може здаватися панацеєю, вона також вносить певний рівень складності. Інтеграція різних мікрофронтендів у єдину систему може вимагати додаткової координації, особливо коли розглядається взаємодія між різними командами та їхніми CI/CD процесами. Використання CI/CD у мікрофронтендах також підкреслює значення моніторингу та журналювання. Здатність відслідковувати зміни, які впроваджуються в систему, і швидко реагувати на будь-які проблеми, що виникають, може бути вирішальним фактором у підтримці стабільності та надійності великих застосунків. В кінцевому підсумку, CI/CD у контексті мікрофронтендів представляє собою могутній інструмент для підвищення продуктивності та якості розробки. Він пропонує новий рівень гнучкості та

контролю, дозволяючи організаціям швидко адаптуватися до змінюваних умов ринку.

Перевірка продуктивності та оптимізація є іншим важливим аспектом. Оскільки кожен мікрофронтенд може мати свої власні ресурси, існує ризик перевантаження клієнтської сторони, якщо не використовувати такі техніки, як ліниве завантаження чи оптимізація ресурсів. Безпека є ще однією областю, яка потребує особливої уваги. З інтеграцією різних мікрофронтендів виникає потреба в узгодженні стратегій аутентифікації, авторизації та інших аспектів безпеки.

Управління станом може також стати складнішим, оскільки додаток розділений на декілька мікрофронтендів. Це може призвести до виникнення проблем з синхронізацією даних між різними частинами застосунку.

Виклик єдності дизайну також може виникнути, коли різні команди розробляють свої мікрофронтенди. Це може призвести до неконсистентності у візуальному відображенні та користувацькому досвіді.

В усьому цьому ключем є планування, комунікація та координація між командами. З правильним підходом та інструментами, багато з цих викликів можна подолати, дозволяючи організаціям отримувати всі переваги мікрофронтендів без значних недоліків.

2.4 Способи комунікації між мікрофронтендами.

У світі сучасних веб-застосунків, де мікросервіси вже стали нормою для бекенду, мікрофронтенди набирають популярності як відповідь на потреби фронтенду. Ця архітектурна парадигма розбиває фронтенд застосунку на менші, незалежні частини. Проте одним з ключових викликів є питання ефективної комунікації між цими частинами.

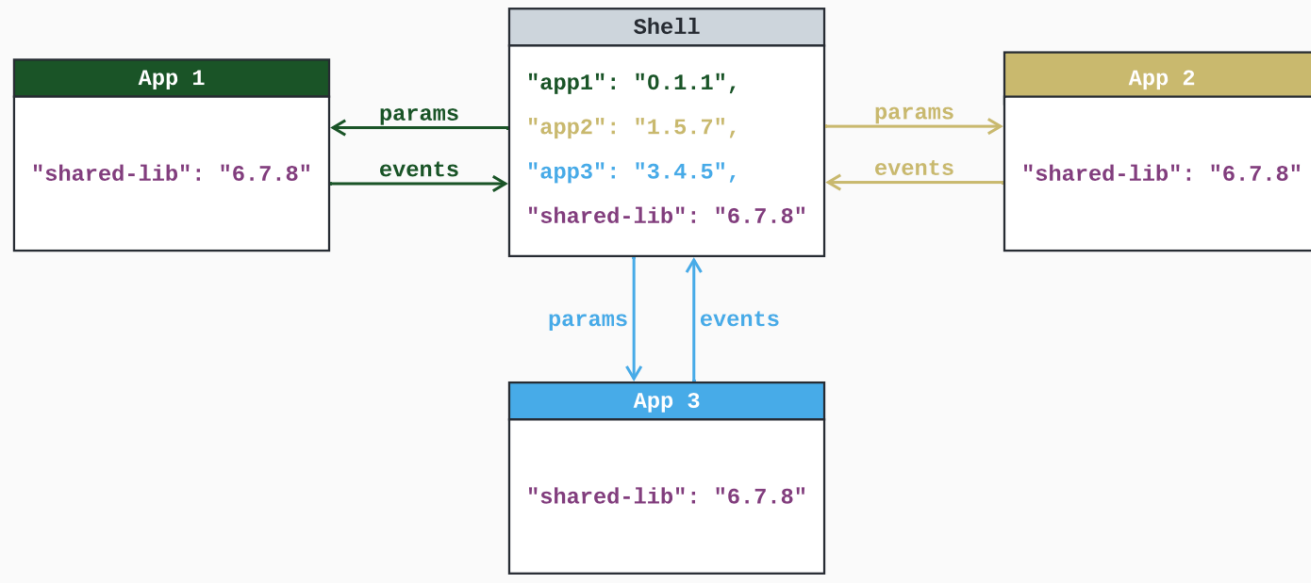

Рисунок — 7 Комунікація між мікрофронтендами

Спільний стан як механізм комунікації виник на початкових етапах розвитку клієнтських застосунків. Ідея полягала в створенні глобального об'єкта, доступного для всього застосунку. Це забезпечувало простоту і прямоту, але при рості застосунку призводило до великої взаємозалежності між компонентами. У великих системах це може призвести до неочікуваних збоїв, коли зміна в одній частині системи впливає на інші частини.

Події стали популярними в архітектурах, заснованих на відгуках. У такому підході, мікрофронтенди публікують і слухають події, створюючи децентралізовану систему комунікації. Це зменшує прямі залежності між компонентами, але вимагає більше уваги до управління потоком подій. Існує ризик "шуму" у системі, де велика кількість подій може ускладнити відладку та розуміння взаємодій.

Виклики API пропонують добре визначений контракт між мікрофронтендами. Кожен мікрофронтенд надає інтерфейс для інших, що дозволяє запитувати дані або ініціювати дії. Це може бути особливо корисно для

синхронних взаємодій. Проте цей підхід може збільшити кількість мережевих запитів і вимагає строгого дотримання контрактів для забезпечення стабільності.

Коли мова йде про вибір методу комунікації, важливо розуміти контекст вашого застосунку та команди. Для невеликих застосунків, де команда розробників невелика, можливо, спільний стан буде достатнім. Для більших, масштабованих систем, де різні команди можуть працювати над різними мікрофронтендами, події або API можуть бути кращим вибором.

Також варто враховувати технологічний стек та існуючі знання команди. Деякі фреймворки або бібліотеки можуть мати вбудовані засоби для певних способів комунікації.

У підсумку, вибір способу комунікації між мікрофронтендами — це важливе рішення, яке може вплинути на масштабованість, продуктивність та стабільність вашого застосунку.

2.4.1 Використання Backend for Frontend для оптимізації комунікації

У сучасному світі веб-розробки виникає все більше потреб у гнучких та ефективних способах інтеграції різних частин великих систем. Один із відповідей на ці виклики є концепція "Backend for Frontend" (BFF). Ця архітектурна ідея, на перший погляд проста, приховує за собою глибокий потенціал для оптимізації взаємодії між frontend та backend. BFF виходить за рамки традиційного розділення на frontend і backend, пропонуючи додатковий рівень, який спеціально розроблений для підтримки конкретного користувацького інтерфейсу. Цей рівень дозволяє інтегрувати декілька джерел даних, сервісів та систем в єдиний API, який відповідає конкретним потребам фронтенду.

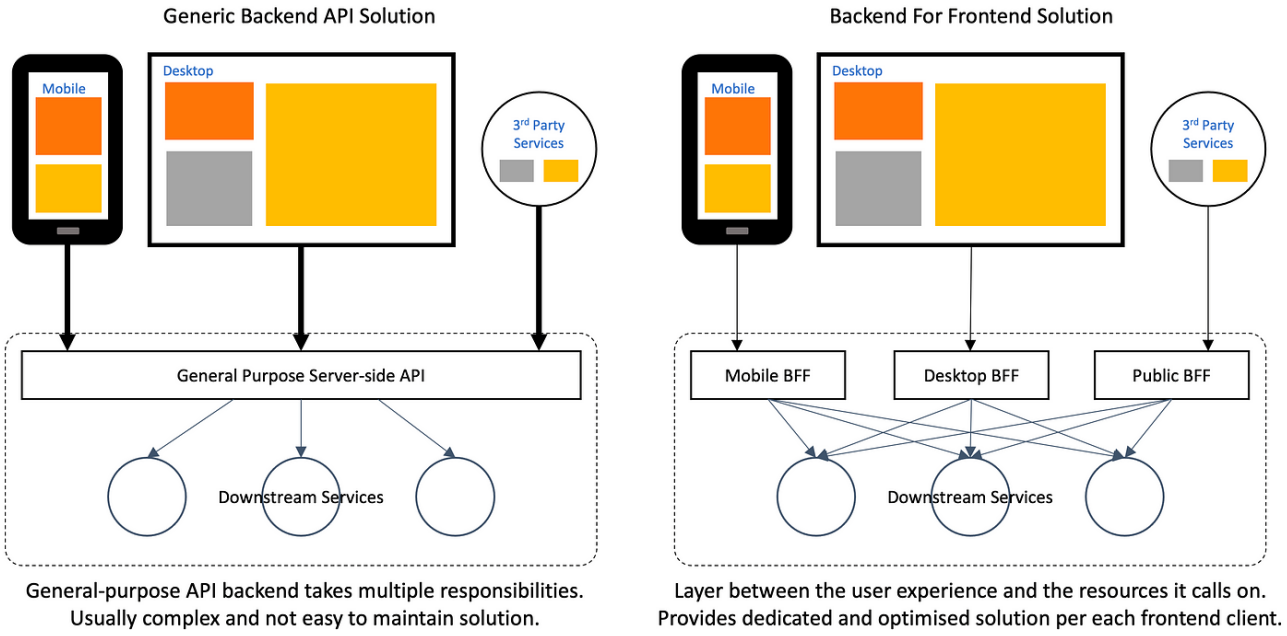

Рисунок — 8 Backend for frontend діаграма

Вивчаючи плюси такої архітектури, можна відзначити, що BFF дозволяє створювати більш гнучкі та оптимізовані API для конкретних потреб користувацьких інтерфейсів. Це, у свою чергу, може призвести до покращення продуктивності, зменшення завантаження мережі та збільшення швидкості відгуку застосунку. Крім того, така архітектура може полегшити процес розробки, оскільки команди можуть фокусуватися на розробці специфічних рішень для конкретних користувацьких потреб. Однак, як і будь-яка архітектурна ідея, BFF має свої виклики та певні недоліки. Введення додаткового рівня може скомплікувати архітектуру системи, збільшити кількість точок відмови та вимагати додаткових зусиль для підтримки. Існує ризик перевантаження BFF завданнями, які могли б бути вирішені на інших рівнях архітектури.

Якщо продовжувати роздуми над концепцією Backend for Frontend, варто звернути увагу на декілька додаткових моментів. У контексті складних та розподілених систем, де множина сервісів спілкуються між собою, BFF може виступати як ефективний "оркестратор". Оркеструючи запити до різних сервісів, BFF дозволяє агрегувати дані, виконувати трансформації та оптимізувати відгук

для конкретного користувацького інтерфейсу. Це може бути особливо корисно у випадках, коли додаток потребує інформації з декількох незалежних джерел.

Тим не менше, впровадження BFF вимагає глибокого розуміння бізнес-логіки та потреб користувача. Погрозою є "силуетування" бізнес-логіки в BFF, коли частина логіки, яка повинна була б бути в сервісах, переміщується до BFF. Це може призвести до того, що BFF стає "товстим", втрачаючи свою первинну роль ефективного агрегатора. Додатково, вибір правильних інструментів та технологій для реалізації BFF є критично важливим. Від цього вибору може залежати продуктивність, масштабованість та надійність системи. Коли ми глибше заглиблюємося в сутність цього підходу, стає зрозуміло, що BFF може служити мостом між монолітними архітектурами та мікросервісами. У традиційних монолітних системах вся бізнес-логіка та представлення даних об'єднуються в єдиному застосунку. Натомість в мікросервісах кожен сервіс є автономним і фокусується на конкретній бізнес-функції. BFF може служити сполучним звеном, яке забезпечує гармонійну взаємодію між цими двома світами, враховуючи найкраще з обох підходів. Також варто розглянути вплив BFF на досвід роботи команди розробників. Через свою специфічність і фокус на конкретних користувачів, BFF може допомогти командам краще розуміти потреби своїх користувачів. Це може сприяти більш ефективній комунікації між командами, які працюють над різними частинами системи, і сприяти кращому розумінню бізнес-логіки. Проте, як і будь-яка інша архітектурна концепція, BFF приносить і свої виклики. Один з них - ризик "дублювання" логіки. Якщо декілька BFF-сервісів потребують однакової бізнес-логіки, існує ризик, що ця логіка буде реалізована в кожному з них окремо. Це може призвести до зайвих витрат ресурсів та ускладнення підтримки.

В узагальненому вигляді, концепція BFF показує великий потенціал для сучасних веб-застосунків, пропонуючи гнучкіший підхід до інтеграції frontend та backend. Однак, як і будь-яке архітектурне рішення, вона вимагає ретельного

аналізу, планування та впровадження, щоб максимізувати її переваги та уникнути можливих пасток.

2.5 Оцінка безпеки мікрофронтендних архітектур

Мікрофронтенди, як інноваційний підхід до розробки веб-застосунків, привертають увагу розробників завдяки своїй гнучкості та модульності. Однак, як і будь-яка нова технологія, вони приносять і свої виклики, зокрема в контексті безпеки. Мікрофронтенди дозволяють розбити великий веб-застосунок на менші, незалежні частини, які можуть бути розроблені, випущені та масштабовані незалежно одна від одної. Цей розподіл може поліпшити безпеку завдяки ізоляції. Якщо один з компонентів має уразливість, він не обов'язково стає слабким місцем для всього застосунку. Однак ця ізоляція також може призвести до викликів. Розробники повинні забезпечити надійну комунікацію між різними мікрофронтендами. Інтеграція між незалежними модулями може створити потенційні точки входу для атак. Крім того, динамічне завантаження коду з різних джерел, що є однією з ключових особливостей мікрофронтендів, може стати ризиком. Якщо завантажений код не проходить належну перевірку, це може призвести до вставки шкідливого коду в застосунок. Та не дивлячись на ці виклики, потенціал мікрофронтендів у контексті безпеки є величезним. Вони пропонують можливість реалізувати додаткові рівні безпеки на рівні кожного компоненту, дозволяючи командам зосередитися на конкретних аспектах безпеки, які є найважливішими для їхньої частини застосунку.Мікрофронтенди, в їх сутності, представляють собою відгалуження від традиційних підходів до веб-розробки, де великі застосунки розбиваються на менші, самостійні компоненти. Ця самостійність надає можливості для детальної адаптації безпеки кожного компоненту до його конкретних потреб. В той час як ізоляція може збільшити стійкість системи до потенційних загроз, вона також може призвести до розриву у загальному розумінні безпеки. У такому контексті, забезпечення

консистентності стає величезним викликом. Кожний мікрофронтенд може використовувати свої власні бібліотеки, залежності та платформи, що може призвести до неконсистентної застосованості політик безпеки. З іншого боку, декентралізований характер мікрофронтендів дозволяє швидко реагувати на потенційні загрози. Якщо в одному з компонентів виявляється уразливість, цей конкретний компонент може бути оновлений або відключений без впливу на інші частини застосунку. Однак ця гнучкість також приносить свої ризики. Коли команди розробників працюють незалежно одна від одної, може виникнути ситуація, коли вони не повністю розуміють, як їхні зміни взаємодіятимуть з іншими частинами системи. Це може призвести до непередбачених прогалин у безпеці.Подальший розгляд мікрофронтендів у контексті безпеки відкриває багатогранний ландшафт, де грані між надійністю, продуктивністю та безпекою стають дедалі більш розмитими.

Важливим аспектом безпеки є автентифікація та авторизація. У системах, де множина мікрофронтендів спілкується між собою та з різними бекенд-сервісами, потреба у єдиному методі ідентифікації користувача стає критичною.

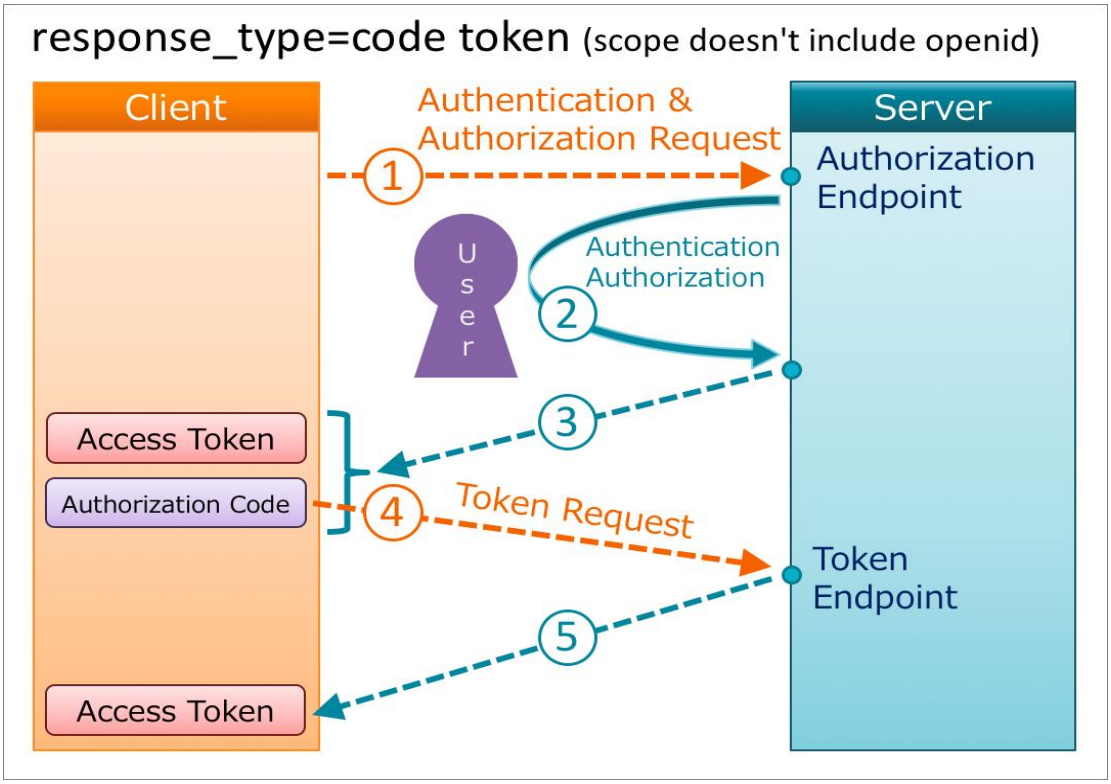

Рисунок — 9 OpenID Connect протокол

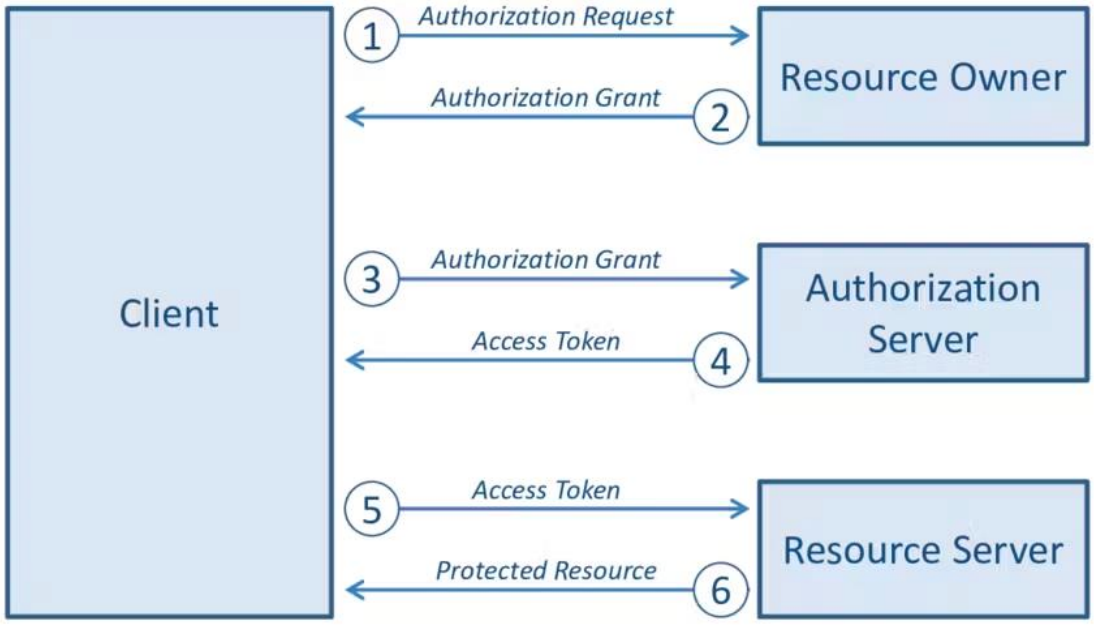

Рисунок — 10. OAuth 2.0 Протокол

Стандарти, такі як OpenID Connect та OAuth 2.0, можуть служити відповіддю на цей виклик, проте їх правильне впровадження та інтеграція у такому розподіленому середовищі вимагає глибокої експертизи. OpenID Connect та OAuth 2.0, два стандарти, що визначають процеси автентифікації та авторизації, з'являються як відповідь на ці виклики.

OpenID Connect, будучи розширенням OAuth 2.0, додає шар автентифікації до останнього, надаючи засоби для користувачів підтвердити свою особистість. Ця додаткова оболонка автентифікації стає особливо цінною у мікрофронтендних системах, де ізоляція компонентів може ускладнювати однозначне визначення користувацької ідентичності.

OAuth 2.0, у свою чергу, створює рамки для того, щоб користувачі могли надавати додаткам дозволи на доступ до їх даних без необхідності розкривати свої облікові дані. У контексті мікрофронтендів це може бути вельми корисно, адже дозволяє ізолювати різні частини системи від непотрібного доступу до користувацької інформації. Проте, як і будь-яка технологія, обидва стандарти мають свої виклики. Їх впровадження вимагає глибокого розуміння протоколів та потенційних ризиків. Необдумане застосування може призвести до витоку даних або незаконного доступу до ресурсів. Тим не менш, потенціал OpenID Connect та OAuth 2.0 для підвищення безпеки мікрофронтендних систем є очевидним. Їхня спроможність ізолювати користувацькі дані та регулювати доступ до ресурсів може служити суттєвим захистом від потенційних загроз. Завдяки гнучкості та модульності мікрофронтендів, команди розробників можуть вибирати, які частини системи мають доступ до конкретних ресурсів, забезпечуючи ефективний контроль безпеки на рівні кожного компоненту.

Паралельно з цим, взаємодія між різними мікрофронтендами може відбуватися через різні домени або походження. Це ставить питання про політику одного походження (Same-Origin Policy) та можливість міжсайтового скриптингу. Ці питання потребують вдумливого розгляду, щоб забезпечити, що

дані користувача залишаються в безпеці. Однією з особливостей мікрофронтендів є їхня динамічність. Ця здатність швидко адаптуватися до змін може бути двостороннім мечем. З одного боку, це дозволяє швидко реагувати на виявлені уразливості, з іншого - може призвести до введення нових уразливостей через недостатнє тестування або аналіз.

2.6 Аналіз та оцінка оптимізації продуктивності мікрофронтендних архіетктур

Оптимізація продуктивності у контексті мікрофронтендів вимагає особливого підходу. Централізовані системи часто користуються перевагами ефективного кешування та оптимізованого завантаження ресурсів, в той час як мікрофронтенди пропонують більш гранульний підхід. Використання мікрофронтендів може допомогти зменшити обсяг завантажуємого коду, завантажуючи лише ті частини застосунку, які необхідні для виконання конкретного завдання. Це може привести до збільшення швидкодії та поліпшення користувацького досвіду, особливо у складних застосунках.

Тим не менше, мікрофронтенди також можуть привести до додаткової складності при інтеграції різних частин застосунку. Наприклад, може виникнути потреба в координації завантаження ресурсів, синхронізації станів або управлінні залежностями між різними мікрофронтендами. Водночас, стратегії оптимізації, такі як ліниве завантаження, дерево стружки коду або оптимізація зображень, можуть бути більш ефективно впроваджені на рівні мікрофронтендів. Це дає можливість командам зосередитися на оптимізації конкретних частин застосунку, замість спроб оптимізувати великий монолітний застосунок. Все згадане вище свідчить про те, що мікрофронтенди пропонують нові перспективи та виклики у контексті оптимізації продуктивності. Їхня унікальна структура та гнучкість можуть допомогти підвищити продуктивність застосунку, але це також вимагає нового підходу до розробки та оптимізації.

Важливим аспектом оптимізації продуктивності в мікрофронтендів є збалансоване управління ресурсами. Хоча мікрофронтенди і надають можливість ізольованого розроблення, вони можуть також привести до перевантаження мережі через множинні взаємодії між компонентами. Зокрема, додаткові HTTP-запити для завантаження окремих мікрофронтендів можуть знизити загальну продуктивність застосунку, якщо це не буде належним чином оптимізовано.

Для вирішення цього питання можна розглянути використання сервіс воркерів або різних стратегій кешування, які можуть прискорити завантаження необхідних ресурсів. Сервіс воркери, наприклад, можуть допомогти в передзавантаженні ресурсів або забезпеченні їх доступності в автономному режимі. Інший важливий аспект - це забезпечення консистентності інтерфейсу користувача. Оскільки різні команди розробляють окремі мікрофронтенди, існує ризик, що стилі та поведінка користувацького інтерфейсу можуть розходитися.

Це може бути вирішено за допомогою спільних бібліотек компонентів та строгих директив щодо дизайну. Не можна також ігнорувати питання безпеки. Ізольована природа мікрофронтендів може призвести до того, що окремі компоненти можуть стати уразливими до атак. Стандарти безпеки та регулярні перевірки є ключовими для забезпечення надійності таких систем.

Процес оптимізації мікрофронтендних систем з'являється як відгук на постійно зростаючі вимоги до продуктивності, швидкодії та користувацького досвіду в сучасних веб-застосунках. З огляду на унікальний характер мікрофронтендних архітектур, необхідно підійти до цього завдання з особливою увагою, враховуючи всі можливі виклики та можливості. Важливим інструментом у нашому арсеналі є бандлери, такі як Webpack та Rollup. Ці інструменти дозволяють не просто структурувати та пакувати код, але й оптимізувати його, видаляючи зайве та забезпечуючи компактність. Водночас, ці бандлери дозволяють гармонійно інтегрувати код, що розробляється різними командами, забезпечуючи цілісність і координованість системи.

Однак, незалежність мікрофронтендів може призвести до певних викликів, зокрема, до потенційного дублювання ресурсів або до складності в управлінні залежностями. Тут на допомогу приходять сервісні працівники, які дозволяють детально керувати завантаженням, кешуванням та використанням ресурсів, забезпечуючи гладке функціонування системи.

Щодо консистентності інтерфейсу та досвіду користувача, слід розглядати створення спільних бібліотек компонентів. Це може бути не просто засобом досягнення однорідності дизайну, а й ефективним способом оптимізації, оскільки спільні компоненти можуть бути завантажені та кешовані один раз для всього застосунку.

Останнє, але не менш важливе, це забезпечення якісного моніторингу та аналітики. Засоби, такі як Lighthouse, можуть допомогти виявити і вирішити проблеми продуктивності, надаючи командам необхідну інформацію для постійного вдосконалення. В узагальненому вигляді, оптимізація мікрофронтендних систем є завданням, що вимагає поєднання правильних інструментів, стратегій та неперервного процесу аналізу. Це завдання, що вимагає глибокого розуміння, але яке може призвести до виняткових результатів у контексті сучасної веб-розробки.

2.6.1 Використання Module Federation для динамічного завантаження модулів

Module Federation — це новаторська технологія, що дозволяє веб-застосункам динамічно завантажувати код на клієнті, і при цьому відсутність необхідності перебудовувати весь застосунок. Ця можливість створює нові горизонти для розробників, дозволяючи їм виготовляти більш гнучкі та адаптивні системи. Основна перевага Module Federation полягає у можливості ділитися залежностями між різними JavaScript-додатками. Це дозволяє розгортати частини застосунку незалежно одна від одної, що є особливо корисним у

контексті мікрофронтендів. Така незалежність може призвести до зменшення часу розгортання та поліпшення продуктивності команди.

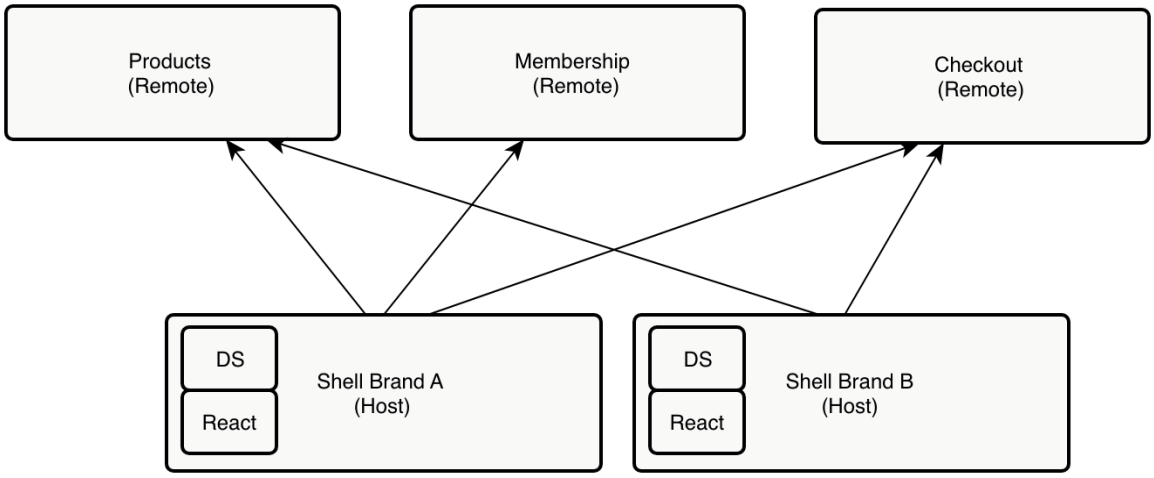

Рисунок — 11 Module Federation діаграма

Проте, як і будь-яка інша технологія, Module Federation має свої виклики. Однією з потенційних труднощів є управління версіями та залежностями між різними частинами застосунку. Щоб уникнути конфліктів, розробники повинні ретельно планувати та координувати оновлення. З іншого боку, Module Federation може призести до покращення користувацького досвіду завдяки меншому часу завантаження. Замість завантаження всього застосунку, користувачі можуть отримати лише ту частину коду, яка їм потрібна у конкретний момент часу. Технічний аспект реалізації Module Federation базується на можливостях сучасних бандлерів, таких як Webpack. Використання таких інструментів дозволяє розробникам створювати конфігурації, що відображають залежності та дозволяють динамічне завантаження модулів. Розглядаючи можливості, які надає Webpack 5 Module Federation, можна визначити, що це не просто засіб для обміну кодом. Ця технологія також дозволяє створювати динамічні бібліотеки компонентів, які можуть незалежно розвиватися і додаватися до відповідних застосунків безпосередньо під час їх виконання. Однак не все так ідеально. Однією з проблем, яка виникає при використанні Webpack 5 Module Federation, є відсутність вбудованих рішень для управління життєвим циклом

мікрофронтендів, а також механізмами роутингу та комунікацій між ними. Це покладає відповідальність за вирішення цих питань безпосередньо на розробника. Проте, важливо зазначити, що Module Federation була створена як інструмент для спільного використання модулів, а не як повноцінний фреймворк для створення мікрофронтендів.

Щодо практичного використання, основна конфігурація Module Federation вимагає визначення залежностей між мікрофронтендами та способів їх динамічного завантаження. Наприклад, конфігурація може вказувати, що основний застосунок залежить від додаткового мікрофронтенду, який буде динамічно завантажений за певною адресою. А сам мікрофронтенд може визначати функції, які він виконує, і які можуть бути викликані головним застосунком.

Цей підхід до динамічного завантаження модулів може відкрити нові можливості для оптимізації завантаження та використання ресурсів, дозволяючи застосунку завантажувати лише ті частини коду, які дійсно потрібні в данний момент.

2.7 Висновки до розділу 2

1. Мікрофронтенди стали ключовим елементом сучасної веб-розробки, дозволяючи командам працювати над окремими частинами фронтенду із забезпеченням незалежності та гнучкості.
2. Один з основних компонентів успіху мікрофронтендів - це їх спроможність до динамічного завантаження, що дозволяє оптимізувати ресурси та покращувати продуктивність застосунків.
3. Webpack Module Federation є важливим інструментом у впровадженні динамічного завантаження, надаючи засоби для шарингу модулів між незалежними застосунками.

4. Незважаючи на гнучкість, яку пропонує Webpack Module Federation, розробники повинні бути готові до вирішення викликів, таких як управління життєвим циклом застосунків, роутинг і комунікація між мікрофронтендами.

5. Застосування динамічного завантаження модулів може стати ключем до створення високопродуктивних веб-застосунків, що ефективно використовують ресурси, завантажуючи лише необхідний в данний момент код.

РОЗДІЛ 3. ПРАКТИЧНЕ ЗАСТОСУВАННЯ МІКРОФРОНТЕНДНОЇ АРХІТЕКТУРИ В HR-ТЕХНОЛОГІЯХ: КЕЙС PERSONIO

3.1 Personio - Огляд та Сфера Діяльності

Personio - це видатний приклад компанії, яка є справжнім лідером у галузі цифрових рішень для управління людськими ресурсами. Ця компанія, яка розташована в серці Європи, пропонує всеохоплюючу платформу для керування всім спектром HR-процесів - від найму та управління талантами до заробітної плати та оцінки ефективності. Ключовою особливістю Personio є її здатність пропонувати інтегровані рішення, що дозволяють організаціям всіх розмірів оптимізувати та автоматизувати ряд складних процесів управління персоналом.

3.1.2 Мікрофронтенди у Контексті Personio

У сучасному світі веб-розробки концепція мікрофронтендів набуває все більшого значення. Вона передбачає розділення фронтенд-частини

застосунків на маленькі, незалежні частини, які можна розвивати, тестувати та розгортати окремо одна від одної. У Personio цей підхід використовується для створення гнучких та легко масштабованих рішень. Використання мікрофронтендної архітектури дозволяє компанії бути більш адаптивною до змін ринкових вимог, підтримувати високий рівень інновацій та ефективно керувати ресурсами розробки.

Мікрофронтенди в Personio відіграють ключову роль у забезпеченні високої продуктивності та гнучкості, дозволяючи командам фокусуватися на конкретних бізнес-задачах та оптимізувати процеси розробки. Цей підхід дозволяє ізолювати різні частини додатку, зменшуючи залежності між командами та спрощуючи процес внесення змін та оновлень.

3.1.3 Значення Мікрофронтендної Архітектури для Personio

Впровадження мікрофронтендної архітектури в Personio є частиною стратегії компанії забезпечити найвищу ефективність та гнучкість розробки. Застосування мікрофронтендів відкриває нові можливості для швидкої адаптації до змінних потреб бізнесу, оптимізації робочих процесів та пропонування інноваційних рішень для управління людськими ресурсами. Це дозволяє Personio не лише підвищити ефективність своїх рішень, але й забезпечувати високий рівень задоволення клієнтів, пропонуючи їм сучасні, швидкісні та надійні сервіси. Мікрофронтенди в Personio є архітектурним підходом, що передбачає розділення великого, монолітного фронтенд-додатку на менші, незалежно розгортовані та управліні частини. Кожна частина представляє собою самодостатній та спеціалізований сегмент, відповідальний за конкретні функції або можливості в рамках загального додатку. Чому Мікрофронтенди ? Існує

кілька причин, чому мікрофронтенди використовуються у розробці фронтенду: Модульність та Розділення Відповідальностей: Мікрофронтенди дозволяють розробникам ділити фронтенд-додаток на менші модулі згідно з бізнес-спроможностями або доменами. Ця модульність сприяє розділенню відповідальностей, дозволяючи командам працювати незалежно над різними областями додатку. Спрощення Розробки та Підтримки: Розділення фронтенд-додатку на менші частини знижує загальну складність системи. Розробники можуть зосереджуватися на логіці специфічної домени, що полегшує розуміння, розробку, тестування та підтримку коду. Незалежне Розгортання та Релізи: Мікрофронтенди дозволяють розгортати та випускати зміни незалежно, без впливу на весь додаток. Ця гнучкість є особливо корисною, коли різні мікрофронтенди керуються різними командами. Покращена Продуктивність та Масштабованість: Мікрофронтенди можуть покращувати продуктивність та масштабованість, використовуючи такі техніки, як ліниве/асинхронне завантаження компонентів. Ізоляція та Толерантність до Помилок: Мікрофронтенди розроблені таким чином, щоб бути ізольованими один від одного. Ця ізоляція допомагає локалізувати проблеми в межах конкретного мікрофронтенду.

Всі додатки в Personio Web слідують однаковій налаштуванню та технологічному стеку, що дозволяє інженерам легко переходити між командами та продуктами. Використання React у Personio Web дозволяє нам створювати додатки модульним чином і вільно обмінюватися компонентами. TypeScript [5] є ще одним очевидним вибором для роботи з JavaScript в Personio Web. Він дозволяє забезпечувати надійність інформації про форму і вхідні дані для функцій та компонентів. Використання Mock

Service Worker (MSW) для імітації мережевих запитів до бекенд-сервісів під час розробки. Тестування в Personio Web включає кілька форм, і кожен метод використовується для перевірки компонентів та функціональності вашого додатку під різними сценаріями та вимогами. Всі додатки в Personio Web збираються та упаковуються за допомогою Webpack. Підтримка Storybook також надається всім додаткам у Personio Web.

Кожна команда несе відповідальність за свій сервіс оркестратора, підтримуючи, вдосконалюючи та розвиваючи потрібні функції.

3.1.4 Структура Фронтенда

Фронтенди в Personio Web мають спільну структуру та налаштування, а також способи використання, чи то в Оркестраторі, чи в Laravel Monolith. Кожен фронтенд містить директорії frontend та integration. Директорія frontend є місцем, де зберігається власне вихідний код додатку. Це основна директорія, в якій ви, швидше за все, будете працювати над розвитком функціональності для вашої команди на щоденній основі. Зміст цієї директорії може дещо відрізнятися залежно від того, як розвивався ваш додаток, але існують деякі основні аспекти, яким повинен відповідати додаток, щоб бути сумісним з Personio Web. Дві найважливіші частини вашого фронтенду - це файл src/application/AppRoot/AppRoot.tsx та файл src/main.tsx.

AppRoot.tsx в Personio Web відіграє роль еквівалента App.jsx з проектів на кшталт Create React App. Він діє як кореневий елемент вашого додатку і саме цей файл буде використано вашою бібліотекою інтеграції фронтенду для рендерингу вашого додатку в Оркестраторі як федеративний модуль. Якщо ваш додаток має специфічні вимоги, наприклад, щодо RUM-часу або вкладених відображень з маршрутизатором, це гарне місце для їх розміщення.

Глобально спільні елементи, такі як SplitFactory та QueryClientProvider, не повинні знаходитися тут. Через федерацію модулів цей файл намагається

"вписатися" у React-дерево Оркестратора та використовувати глобальні налаштування Split та React Query. Додавання власних елементів ізолює ваш фронтенд від цих функцій та, ймовірно, не є бажаною поведінкою.

Main.tsx використовується для побудови вашого фронтенду у спосіб, сумісний зі старою версією, що дозволяє йому бути відтвореним у Laravel monolith. Якщо ви знайомі зі старою самостійною налаштуванням мікрофронтенду, цей файл відтворює це. Він надає конфігурований обгортку навколо вашого AppRoot.tsx, яка включає все необхідне для роботи вашого фронтенду як самостійного елемента.

У Оркестраторі бібліотека інтеграції вашого додатку використовується для рендерингу вашого федеративного модуля (AppRoot.tsx) на сторінці. Оркестратор надає власні копії бібліотек, таких як Split і React Query.

У моноліті main.tsx вашого додатку завантажується з CDN і рендериться на сторінці традиційним способом (за допомогою скрипту та елемента для монтування). main.tsx вашого додатку включає той самий AppRoot.tsx, що й бібліотека інтеграції для Оркестратора, але також містить власні копії бібліотек, таких як Split і React Query. Кожна команда несе відповідальність за свій сервіс оркестратора, підтримуючи, вдосконалюючи та розвиваючи необхідні функції.

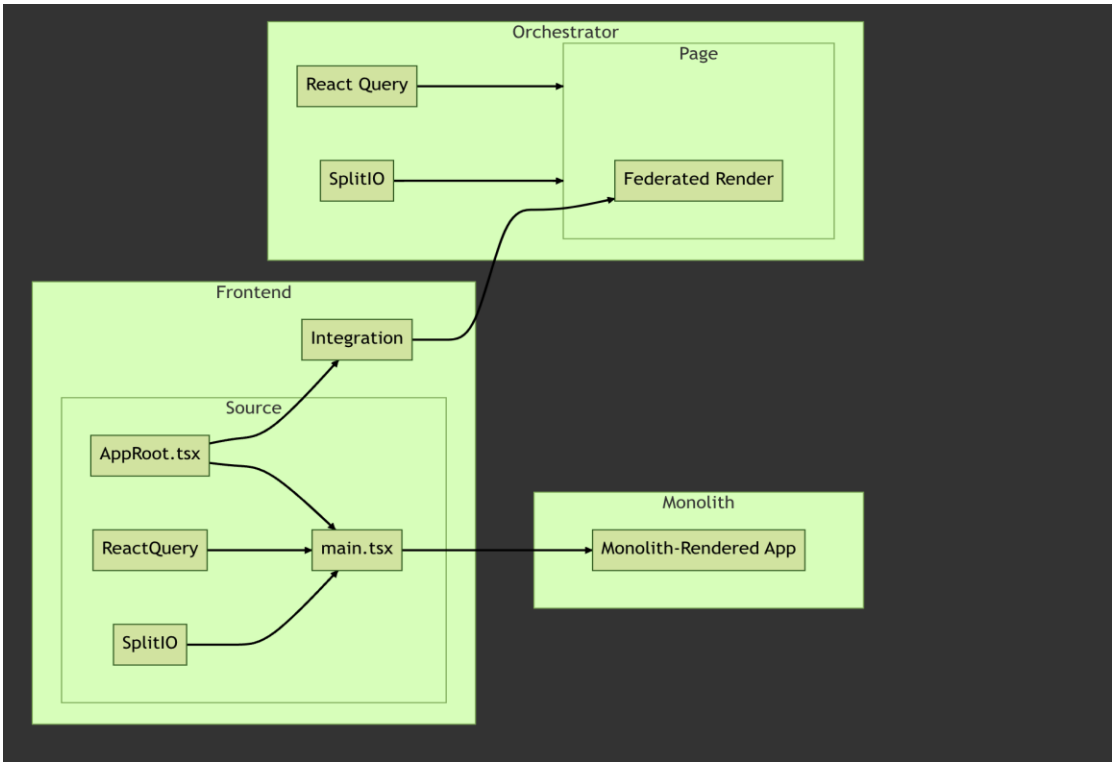

Рисунок — 12 Схема рендеріга фронтенда в web-додатку Personio

3.2 Ааналіз викликів розробки фронтенду в Personio та стратенгії їх вирішення.

3.2.1 Дослідження старого підходу до фронетнду в веб-застосунку Personio

Personio почалася як PHP-додаток, написаний на Laravel, що є MVC фреймворком. У Laravel двигун шаблонів Blade відповідає за "View", який транспілюється у кешований PHP-код. Коли говорять про "blades" або "шаблони" у контексті моноліту, мається на увазі саме це. Для розробки функціональності на певному маршруті, такому як "Профіль Співробітника", багато бізнес-логіки будувалося в Контролерах і передавалося у Вид через Модель. У деяких випадках додаткова логіка вводилася безпосередньо в Blade (двигун шаблонів дозволяв використовувати структури управління, умовні оператори та цикли). Додаткові можливості на сторінці реалізовувалися за допомогою jQuery, що дозволяло забезпечити клієнтську інтерактивність. З часом jQuery вийшов з моди, і з появою клієнтських бібліотек для рендерингу, таких як React, стало легше створювати

багаті клієнтські досвіди. Для використання цих переваг React був введений у моноліт, дозволяючи розробникам створювати додатки на React без необхідності знань jQuery або PHP. Все це існувало всередині монолітного репозиторію (personio), і з кожним розгортанням весь кодовий базис оновлювався і розгортався одночасно. Розгортання ставали довгими, важкими і схильними до помилок. Щоб не натрапити один на одного під час розгортань, співробітники Personio вручну чергували релізи та планували їх у каналі Slack.

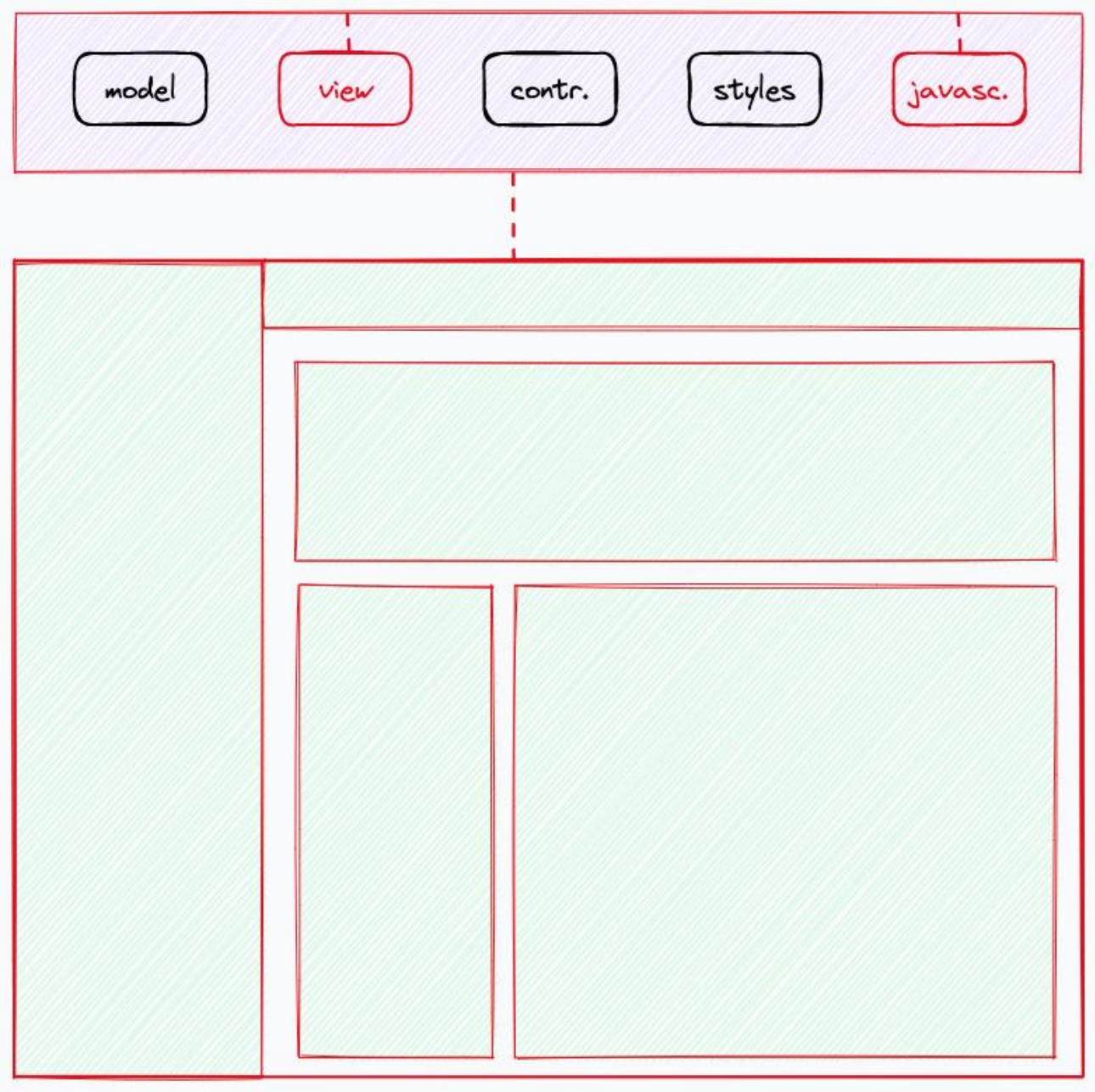

Рисунок — 13 Огляд старої схеми форнтенда Personio

З розвитком Personio як єдиного PHP-додатку, виникали складнощі, особливо з точки зору продуктивності. У міру зростання та розвитку програми, стало дуже легко ненароком знизити продуктивність у всьому додатку. Через специфіку зростання програми було важко визначити конкретну команду, яка могла б взяти на себе відповідальність за ключовий функціонал.

Це створювало додаткові виклики, оскільки кожна частина додатку була взаємопов'язаною та взаємозалежною, що ускладнювало внесення змін та вдосконалень без ризику негативного впливу на інші частини системи. Отже, потреба в новому підході ставала очевидною для забезпечення більшої гнучкості, масштабованості та ефективності в розвитку додатку.

3.2.2 Розгляд інтеграції мікрофронтендів

З метою збільшення швидкості розробки та автономії команд, інженери звернулися до патерну мікрофронтендів (MFE). Цей підхід включає створення додатків React як самостійних застосунків та розгортання статично побудованих JS-бандлів у локацію CDN. Шаблони Blade були оновлені для включення "вузла монтування", кореневого елементу, в який рендериться React-додаток, та додаткових тегів скрипту для завантаження JS-бандлів з CDN. Можна уявити це як Create React App, але замість статичного файлу index.html HTML надається Laravel.

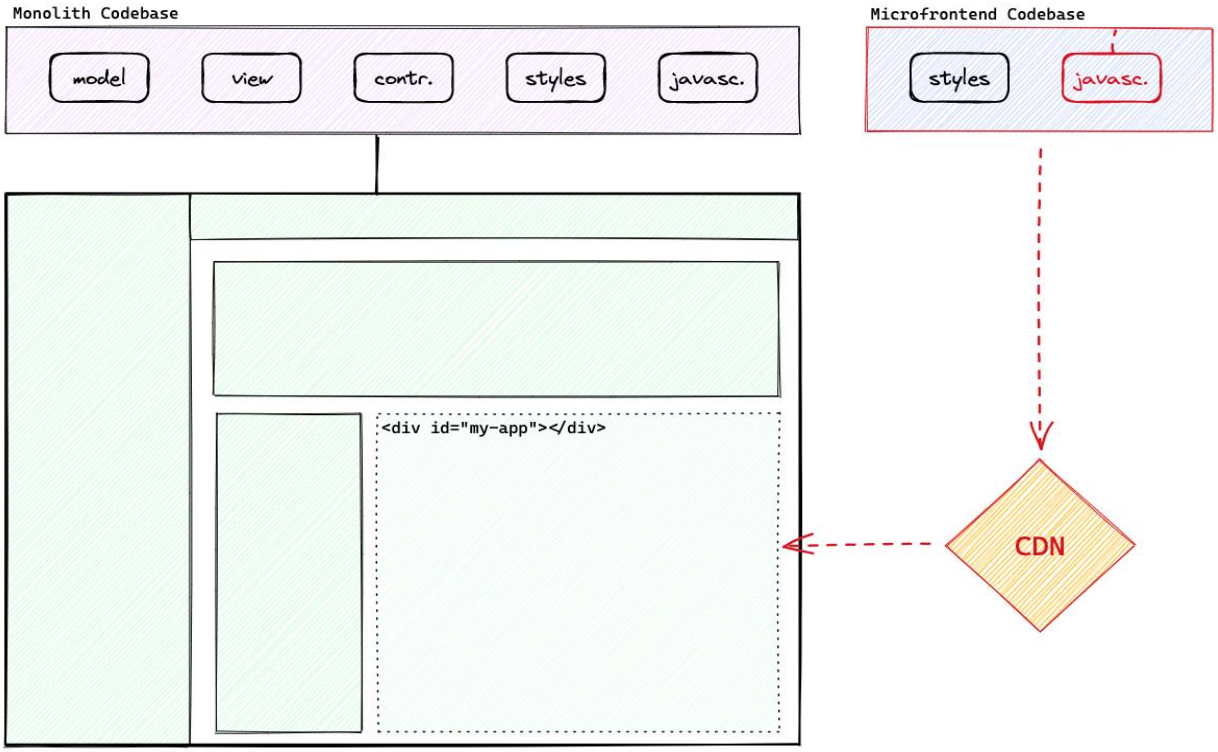

Рисунок — 14 Огляд схеми розробки мікрофронтенда

Для реалізації цього була створена інфраструктурна частина, відома як Служба Artifact. Служба Artifact зберігає "запис" про те, які активи вважаються "активними" для даного застосунку. Під час розгортання додатку в CI робиться запит до Служби Artifact зі списком назв файлів, які синхронізуються з S3. Коли Моноліт звертається до Служби Artifact з інформацією про даний додаток, вона повертає розташування "активних" файлів у S3 для цього застосунку. Ця інформація потім використовується для додавання тегів скрипту та стилю до шаблону Blade. Все це надало більшої автономії фронтенд-командам і дозволило їм повністю відділити життєвий цикл розробки свого додатку від решти Моноліту, сприяючи швидшій розробці функціональності та релізів. Зміни в коді додатку будувалися та розгорталися у власних CI-пайплайнах без необхідності вносити будь-які зміни до Моноліту.

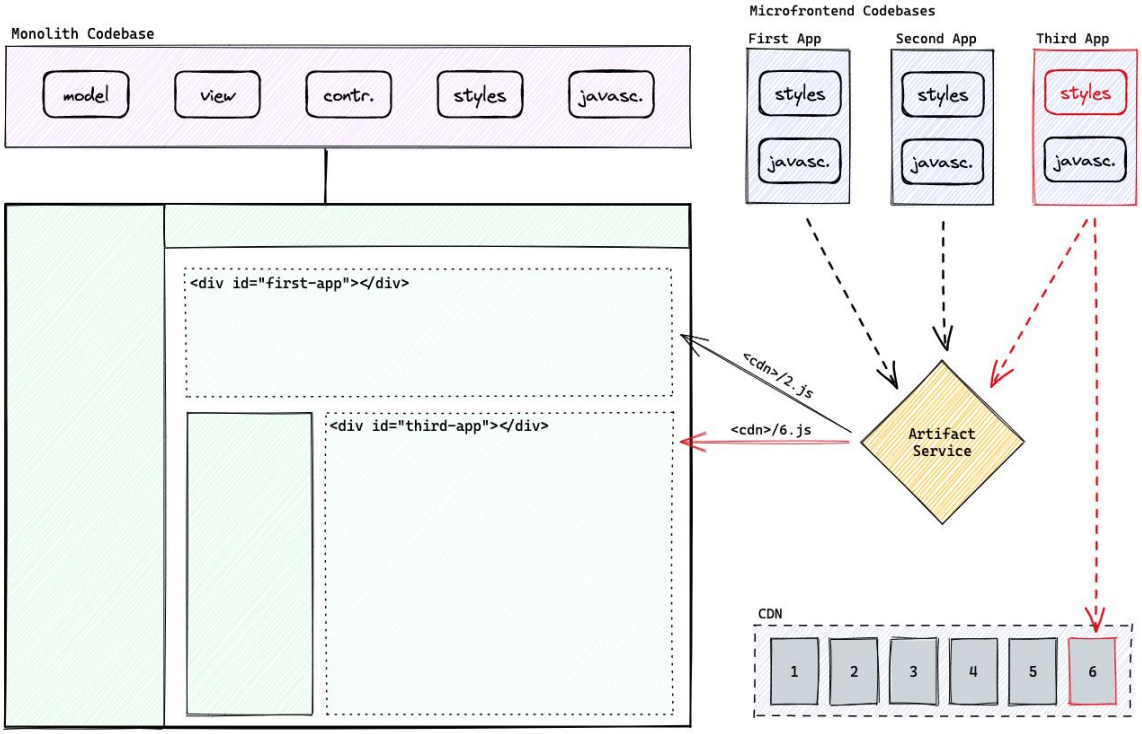

Рисунок — 15 Звернення мікрофронтендів до Artifact Service

Принаймні, так було задумано.Багато команд все ще покладалися на дані, які можна було отримати лише в Моноліті, поки чекали на створення та розробку самостійних мікросервісів. Щоб отримати доступ до цих даних зі свого додатку, багато команд прикріплювали дані до об'єкта window у шаблоні Blade, ненавмисно зв'язуючи свій додаток назад з Монолітом.

Хоча додатки дозволили командам будувати та розгортати свої застосунки швидше, ніж якби вони робили це в Моноліті, це також спонукало команди розробляти свої додатки ізольовано. Часто рішення та допоміжні функції були дубльовані з додатку в додаток.

Щоб вирішити цю проблему, інженери створили кілька загальних бібліотек для спільного використання коду між додатками. Ці бібліотеки часто не мали визначеного відповідального за їх підтримку, замість цього покладаючись на тоді невелику та згуртовану фронтенд-спільноту в Personio. Це часто призводило до оновлення бібліотеки з новими функціональностями однією командою для використання у їх конкретному додатку. Команди оновлювали бібліотеку та свій

додаток, але інші користувачі цієї бібліотеки повинні були самостійно оновити свій додаток до останньої версії.

Додатки, які були залишені без підтримки, або команди, у яких не було багато фронтенд-ресурсів, часто стикалися з необхідністю вирішувати складні проблеми внаслідок великих змін у цих бібліотеках. Часто багато додатків просто існували з застарілими залежностями та бібліотеками. Крім того, через спосіб роботи додатків із активами Моноліту, деякі залежності та бібліотеки були неможливими або дуже складними для безпечного оновлення (наприклад, React, ReactDOM).

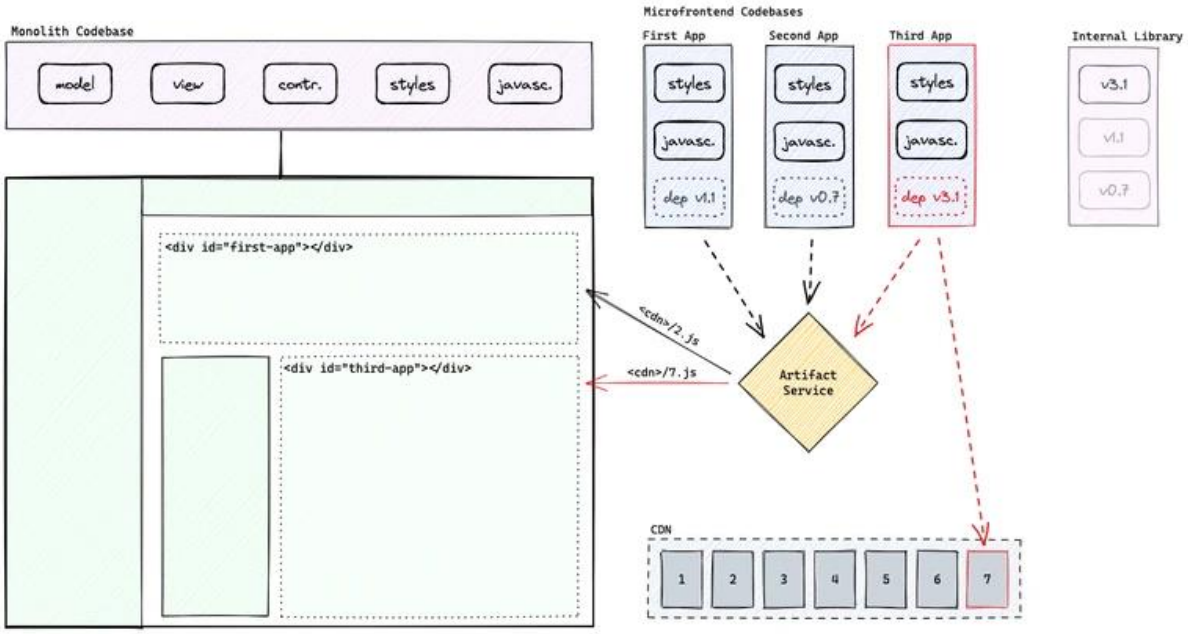

Рисунок — 16 Додання бібліотеки до фронтенду

Однією зі стратегій, використаних при реалізації мікрофронтендів, було застосування опції конфігурації Webpack Externals. З цим підходом великі спільні залежності Personio, такі як React і ReactDOM (серед інших), були видалені з додатків під час побудови, замість цього вони посилалися на версії, що поставлялися через JS-бандл Моноліту. Цей підхід спочатку працював добре і згідно з задумом, однак із відокремленням все більшої кількості додатків від Моноліту стало складніше зберігати повну видимість усіх додатків, які могли бути вплинуті змінами в залежностях на рівні Моноліту.

Це, нарівні з тим фактом, що версія цих бібліотек мала підтримувати код на рівні React, який існує всередині Моноліту, призвело до того, що все більше і більше команд вилучали бібліотеки, на кшталт React, з опції externals своїх додатків і замість цього поставляли свої версії, вбудовані у свій додаток. Неминуче це поступово призводило до збільшення розміру сторінок.

Маючи всю цю інформацію на увазі, важливо зазначити, що додатки все ще вимагають Моноліту для рендерингу "основної" HTML-сторінки та всіх необхідних тегів скриптів і стилів для цього додатку. Знецінення Моноліту передбачає остаточне видалення монолітного додатку, і як таке, нам все ще потрібен спосіб рендерингу цих додатків для наших клієнтів.

Це означає, що попри розвиток і впровадження мікрофронтендів, існує "слон в кімнаті" - залежність від Моноліту як основи для рендерингу додатків. Зниження ролі та відмова від Моноліту є важливою частиною стратегії розвитку, але водночас це створює виклики для забезпечення стабільності та доступності додатків в перехідний період. Таким чином, інженерам та командам розробників потрібно знаходити баланс між інноваціями та надійністю, впроваджуючи нові підходи, не втрачаючи з ока важливість стабільного і безперебійного надання послуг клієнтам.

3.2.3 Оцінка та впровадження фронтенду нового покоління

З метою розвитку та впровадження нової платформи рендерингу для додатків, фронтенд-спільнота розглядала різні підходи для запуску цього процесу. Після тривалих обговорень переваг та недоліків різних стратегій, інженери вирішили обрати підхід, що найкраще відповідає їх потребам. Вибір критеріїв та вагомості кожного з них було визначено за допомогою процесу Dot Voting.

Під час інженерного хакатону 2021 року команда інженерів розпочала створення компанійського монорепозиторію. У цей період було розроблено концепцію оркестраційного шару на базі Next.js, яка була інтегрована у

монорепозиторій. Використання Next.js як основного механізму рендерингу для додатків дозволяє обмінюватися кодом у всьому стеку та надавати готові рішення для загальних сценаріїв, зустрічаючихся у додатках. У міру прогресу хакатону стало очевидно, що потужність федерації модулів відмінно доповнює оркестраційний шар у контексті індивідуально розгорнутих додатків. Поєднання цих двох елементів дозволяє оркестраційному шару та додаткам, що рендеряться всередині нього, функціонувати так, ніби вони створені для одного єдиного додатку. Додатки залишаються незалежними з власними процесами побудови, тестування та розгортання. Інтеграція досягається через різні точки входу в додатки.

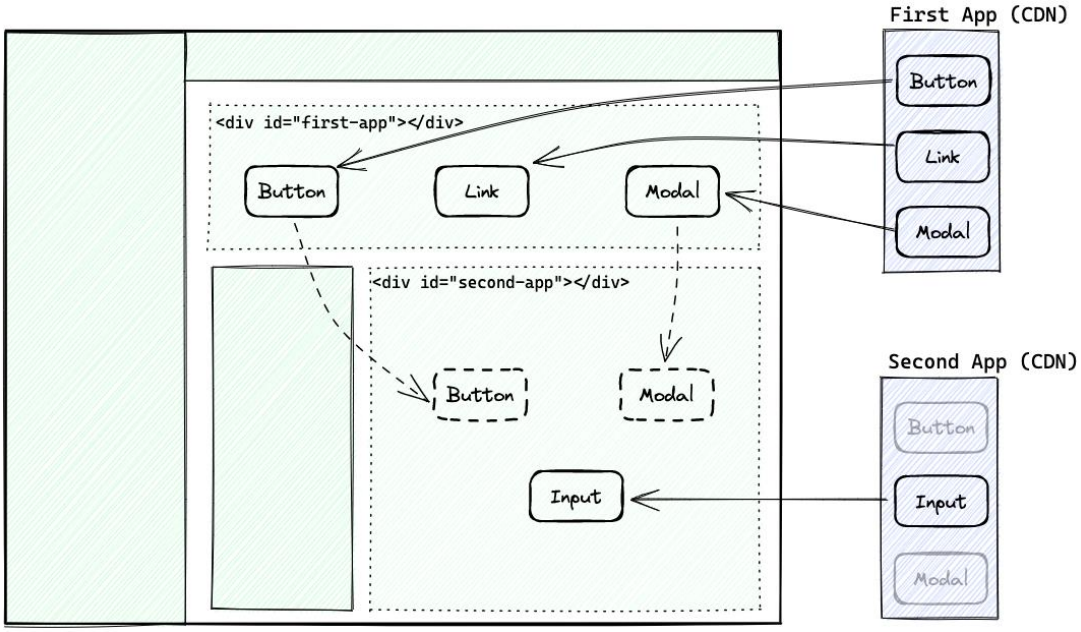

Рисунок — 17 Схема фронтенду нового покоління

Одна з точок входу дозволяє додатку функціонувати в "standalone mode" (самостійному режимі), що дає можливість командам продовжувати рендеринг своїх додатків так, як вони традиційно робили це в Моноліті. Ця точка входу містить усі обгортки, які можуть знадобитися додатку для самостійної роботи (ReactQuery, Split, Sentry) та розгортає повний пакет на CDN, який рендериться в Моноліті за допомогою тегів скрипту. Під час процесу побудови додатки

розділяються на дві частини у "точці розриву". Від цієї точки розриву можна завантажити підрозділ додатку безпосередньо в Оркестратор за допомогою Федерації Модулів. Це точка входу для Федерації Модулів. Точка входу Федерації Модулів використовується всередині точки входу для самостійного режиму, але про це пізніше.

Це підхід дозволяє гнучко розмежувати додатки, забезпечуючи їх взаємодію із загальною інфраструктурою та одночасно зберігаючи можливість самостійного рендерингу. Використання Федерації Модулів для вбудовування підрозділів додатків безпосередньо в Оркестратор відкриває нові можливості для гнучкого та ефективного управління взаємодією між різними частинами додатків і платформи.

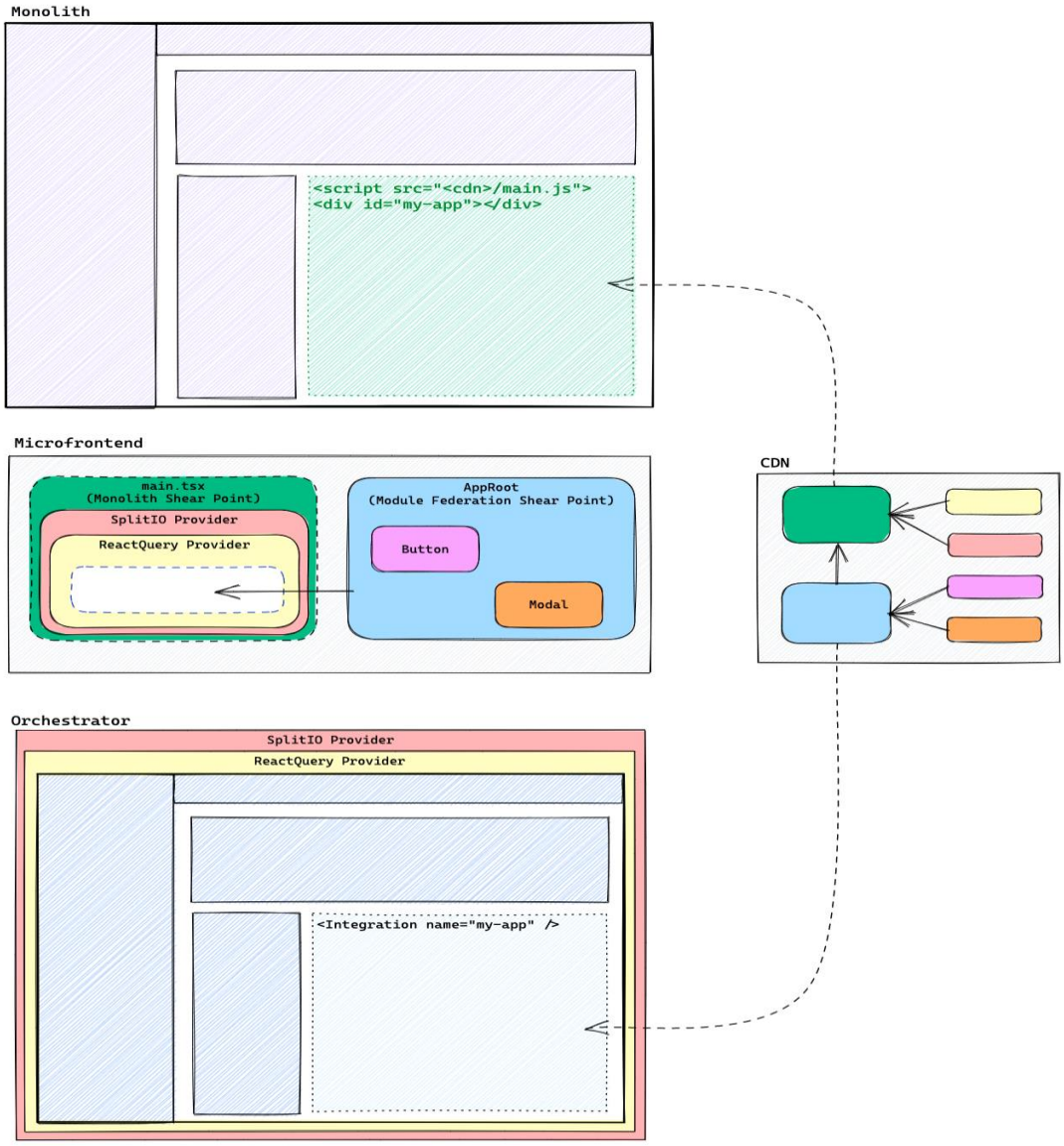

Рисунок — 18 Анатомія фронтенда у веб-додатку Personio

Оскільки додатки на сторінці працюють всередині Оркестратора, ніби вони є частиною одного великого монолітичного застосунку, вони можуть використовувати інстанції бібліотек та провайдерів Оркестратора, таких як ReactQuery, Split, Sentry. Це дозволяє розподіляти спільні активи по всьому фронтенд-стеку та допомагає зменшувати загальний розмір пакету, коли користувачі переходять по різних частинах додатку.

Такий підхід сприяє ефективній оптимізації ресурсів, оскільки не потрібно завантажувати одні й ті ж бібліотеки для кожного додатку окремо. Відтак,

користувачі отримують більш швидкий та плавний досвід користування додатками, а розробники мають змогу зосередитися на унікальних аспектах кожного додатку, не переймаючись про загальні залежності.

3.2.4 Розбір інструментів та підходів в розробці фронтенду нового покоління

Для найкращого використання нових функціональних можливостей, які могли запропонувати новітні фронтенди, було важливо зберегти узгодженість версій залежностей і спростити обмін спільними рішеннями між фронтендами, тому було введено монорепозиторій Personio Web.

Монорепозиторій — це підхід до розробки програмного забезпечення, при якому весь код різних проектів зберігається в одному репозиторії. Термін "моно" відноситься до того факту, що весь код знаходиться в одному місці, а "репо" означає репозиторій. Цей репозиторій містить увесь код різних фронтенд-додатків та бібліотек, над якими працює Personio.

Одна з основних переваг монорепозиторію — можливість спільного використання бібліотек та залежностей. Коли весь код знаходиться в одному репозиторії, легко повторно використовувати код між проектами. Це може призвести до значної економії часу, оскільки розробникам не потрібно переписувати код, який вже був написаний. Це також може зменшити кількість помилок, оскільки код, який був протестований і перевірений, можна повторно використовувати без необхідності розширеного повторного тестування. Узгодження версій залежностей також значно полегшується, оскільки оновлення, включаючи розв'язання потенційно критичних змін, можуть бути виконані одночасно для всіх фронтенд-додатків.

Ще одна перевага монорепозиторію — це можливість спільного використання конвеєрів і етапів. Оскільки весь код знаходиться в одному репозиторії, легко налаштувати конвеєр безперервної інтеграції та доставки (CI/CD), який працює для всіх фронтенд-проектів в Personio. Це означає, що

розробники можуть легко будувати, тестувати та розгортати свій код і робити це без необхідності турбуватися про тонкощі роботи з кількома репозиторіями.

Одним з ключових аспектів монорепозиторію є власність коду. Розподілена, але зосереджена власність є обов'язковою. Кожна команда може володіти своїм конкретним додатком і бібліотеками. Спільна власність основних файлів додатків та бібліотек (наприклад, package.json, tsconfig.json) дозволяє швидко вносити глобальні зміни, що сприяє забезпеченню узгодженості по всьому кодовому базису.

Розробники мають в своєму розпорядженні кілька інструментів під час роботи всередині Personio Web. Всі додатки створюються за допомогою генератора мікрофронтендів, який встановлює базовий додаток для команд. Ця базова конфігурація надає кілька корисних інструментів, таких як інтеграція I18N, імітація API, конфігурація та провайдер React Query, налаштування Split для функціонального флагування, а також усі необхідні скрипти для тестування, побудови та розгортання, які потрібні для роботи в монорепозиторії.

Крім того, команди мають повний доступ до імітації API через MSW, яка має безшовну інтеграцію з локальним набором інструментів для розробки. Це дозволяє розробникам працювати з макетними обробниками "на льоту" безпосередньо в браузері, маніпулюючи з ними та адаптуючи до потреб розробки.

Ці інструменти сприяють ефективній роботі в монорепозиторії, дозволяючи командам швидко та ефективно розробляти та тестувати свої додатки. Використання загальних інструментів і бібліотек спрощує процес розробки та сприяє забезпеченню узгодженості і якості додатків по всьому фронтенд-стеку.

Усі новостворені додатки живуть і генеруються всередині Personio Web, причому багато легасі-додатків вже мігрували або перебувають у процесі міграції. Усі MFE в Personio Web мають інтеграційну бібліотеку, яка існує поруч

з їхнім додатком. Ця бібліотека використовується для безпосередньої інтеграції додатку в Оркестратор (або інші додатки) за допомогою Федерації Модулів та обробляється прозоро внутрішніми бібліотеками та інструментами, абстрагуючи від розробників. Під капотом, інтеграційна бібліотека просто вказує на розгорнутий Федеративний Модуль, який команди визначили у своїй конфігурації додатків і розгорнули на CDN. Цей модуль є лише невеликою частиною їхнього додатку, яку вони визначили для сторінки, і не містить жодних додаткових бібліотек чи обгорток/провайдерів, які можна знайти у їх основному файлі.

Завдяки Оркестратору додатки залишаються відокремленими від процесу розробки та розгортання Оркестратора і можуть продовжувати будуватися та розгортатися окремо. Будь-які оновлення додатків поширюються на CDN, де Федерація Модулів запитує та використовує останні файли в режимі реального часу.

Звісно, не всі додатки в Personio Web рендеряться в Оркестраторі. Більшість додатків наразі рендеряться у Моноліті і таким чином не мають можливості спільного використання глобального контексту через Федерацію Модулів, як це роблять в Оркестраторі.

Для цього всі додатки використовують допоміжний утиліт initMFE в Personio Web. Цей інструмент бере точку входу Федерації Модулів з додатку і обгортає її всіма необхідними провайдерами React, утилітами та конфігураціями, необхідними для роботи додатку в самостійному режимі. Це потім будується та розгортається на CDN, де може бути використане традиційним способом за допомогою тегів скрипту, які надає Служба Artifact, та вузла монтування в HTML, який надає Моноліт.Це означає, що додаток, коли він рендериться в Оркестраторі, і додаток, коли він рендериться в Моноліті, є одним і тим же додатком, який використовує ті ж внутрішні компоненти, лише з додатковими обгортками у версії Моноліту.

Об’єднуючи все разом ми маємо наступну картину поточної стратегії розробки фронтенда: Команди, які генерують додаток в Personio Web, на 99% готові до інтеграції їхнього додатку в Оркестратор. Усі додатки, створені всередині Personio Web, слідують очікуваній структурі, яка дозволяє Федерації Модулів працювати, а також генерувати самостійний пакет для їхнього додатку для рендерингу в легасі-режимі в Моноліті. Команди з легасі-репозиторіями самостійних MFE-додатків повинні зробити деяку роботу для міграції свого додатку в Personio Web таким чином, щоб використовувати наявні інструменти та бібліотеки. Це зазвичай включає генерацію нового додатку в Personio Web та поступову міграцію їхнього існуючого функціоналу зі старих додатків.

Таким чином розвиток фронтенд-платформи в Personio Web є значним кроком у напрямку модернізації та оптимізації процесу розробки. Впровадження монорепозиторію та інтеграція мікрофронтендів через Федерацію Модулів та Оркестратор відкривають нові можливості для більш гнучкої, ефективної та незалежної розробки. Це дозволяє командам бути більш автономними та швидко адаптуватися до змін, забезпечуючи при цьому консистентність і якість кінцевого продукту. Використання монорепозиторію сприяє легшому управлінню залежностями, спільним бібліотекам та інструментам, а також забезпечує ефективне використання спільних ресурсів. Це також уможливлює спільну роботу над проектами, покращуючи взаємодію між командами та сприяючи швидкій інтеграції нових функцій та оновлень.

Мікрофронтенди та Федерація Модулів відкривають шлях для більш модульного та гнучкого підходу до розробки. Це дозволяє кожному додатку залишатися незалежним і легко інтегрованим в ширший контекст платформи, забезпечуючи при цьому консистентний та ефективний користувацький досвід.

Нарешті, впровадження ініціативи по інтеграції в Оркестратор та підтримка існуючих додатків у Моноліті забезпечують гладкий перехід та продовжують підтримувати стабільність у фазі міграції. Це гарантує, що незалежно від того, де

рендериться додаток - у Оркестраторі чи Моноліті - він забезпечує однакову функціональність та якість. Усе це свідчить про стратегічний підхід Personio до інновацій у фронтенд-розробці, забезпечуючи стійку платформу для зростання та адаптації до майбутніх викликів у галузі.

3.2.5 Огляд служби Artifact

Служба Artifact є важливою частиною інфраструктури фронтенду в Personio, відповідаючи за відповідність між кожним маршрутом, який відвідує кінцевий користувач в веб-додатку Personio (наприклад, "/staff"), та JS, CSS та іншими активами, необхідними для належної роботи відповідного мікрофронтенду (наприклад, мікрофронтенду списку співробітників). Служба Artifact розроблена на Kotlin та працює на JVM.

Служба Artifact працює як окрема служба в кластері та обслуговує трафік, який надходить з Моноліту. Вона не обслуговує трафік з публічної мережі. Вона виконує дві задачі: Встановлює відповідність між кожним маршрутом (шаблоном URL-шляху), який відвідує кінцевий користувач у додатку Personio, з активами, що мають відношення до мікрофронтенду, призначеного для роботи на цьому маршруті. Оновлює відображення маршруту до активів за запитом, наприклад, коли розгортається нова версія мікрофронтенду.

Служба Artifact не "знає" про жоден з мікрофронтендів. Насправді, вона навіть не гарантує, що активи, пов'язані з будь-яким з мікрофронтендів, існують та доступні за URL-адресами, які вона надає — це є відповідальністю кожного з мікрофронтендів. Якщо такий URL існує, але веде в нікуди, чи то через видалення файлу, чи через початково неправильно сформований URL, служба Artifact продовжуватиме надавати цей URL без будь-якої перевірки. Вона також не "каже" браузеру кінцевого користувача, які URL завантажувати. Натомість, це є відповідальністю спільного мікрофронтенд-рантайму, який наразі представлений Монолітом і який забезпечує, що браузер кінцевого користувача отримує всі

відповідні URL-адреси та завантажує їх для того, щоб завантажити та виконати активи.

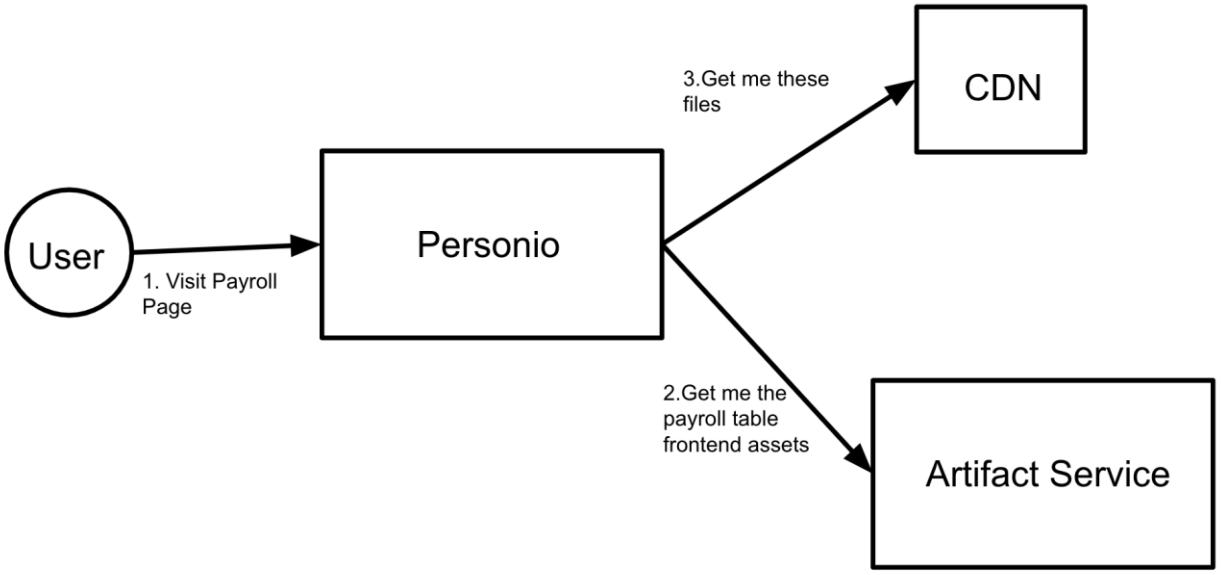

Рисунок — 19 Схема отримання ресурсів

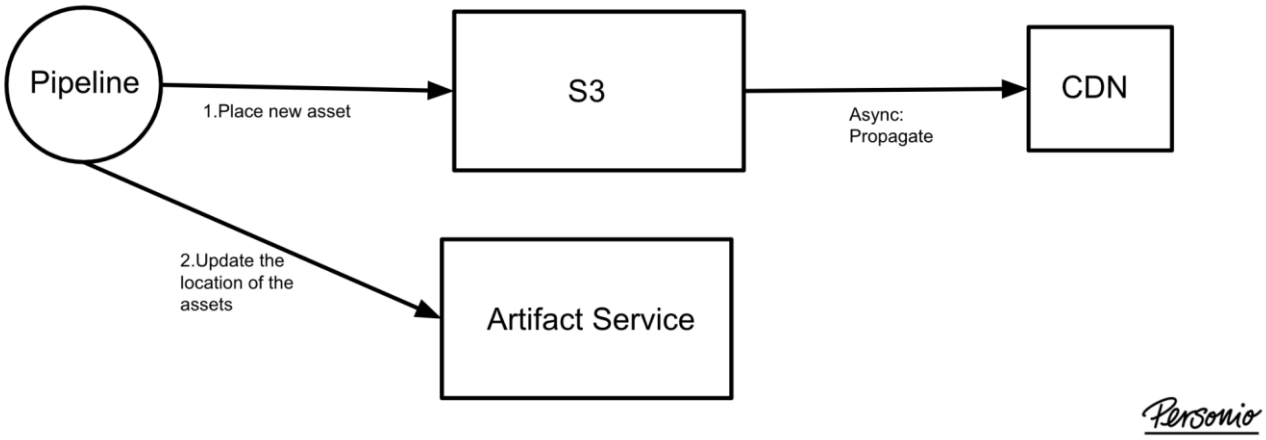

Рисунок — 20 Схема розгортання служби Artifact

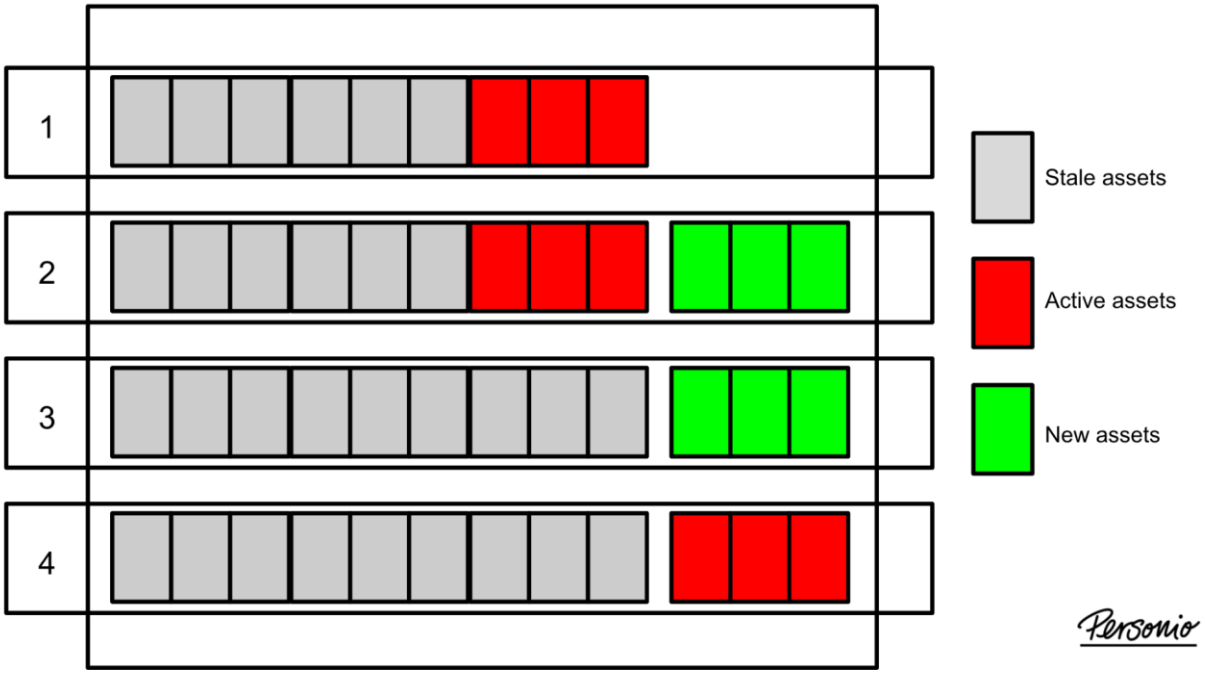

Рисунок — 21 Схема додання нових ресуріс до служби Artifact

3.4 Детальний аналіз технології орекстратора

Фронтенд Оркестратор в Personio - це додаток на базі Next.js, відповідальний за створення основного макету сторінки за моделлю AppShell. Цей AppShell містить базову HTML-структуру для компонування сторінки та забезпечує всі перегляди навігацією та верхньою панеллю. Цей документ не зосереджується на тонкощах Next.js, а сконцентрований на специфіці використання Next.js в Оркестраторі Personio. Для детальнішої інформації про Next.js рекомендуємо ознайомитися з офіційною документацією.

Next.js - це веб-фреймворк на основі React, який підтримує TypeScript, і таким чином є тим самим концептуальним фреймворком, в якому існує фронтенд-робота Personio. Використання React як бібліотеки рендерингу та однієї мови дозволяє нам зосередитися на розвитку служби та спрощує зусилля команд для інтеграції з нею, усуваючи потребу вивчати нові парадигми.

Завдяки використанню React в Next.js як мови шаблонів, ми можемо створити єдиний контекст рендерингу для всіх внутрішніх фронтенд-додатків,

які рендеряться у його виді. Це означає, що ми можемо мати один екземпляр бібліотек і конфігурацій, що обгортають весь додаток, таких як React Query, Sentry та Datadog, без необхідності створювати та перенастроювати їх для кожного додатку.

Next.js дозволяє нам створити власний сервер, який обслуговує всі запити до та від служби Next.js. Різні бізнес-критичні функції відбуваються всередині цього власного сервера (який ми побудували на Express), такі як отримання/оновлення JWKS та декодування JWT, шар сумісності для специфічної поведінки моноліту (який поступово виводиться з ужитку), та ініціалізація серверного трасування Datadog. Next.js також підтримує рендеринг на стороні сервера як спосіб поліпшення першого досвіду завантаження сторінки для кінцевого користувача. Використовуючи інтеграційну бібліотеку, команди можуть доставляти стан завантаження для свого додатку на сервер для розміщення на критичному шляху рендерингу. Це дозволяє Оркестратору намалювати базову форму додатку, не переносячи аспекти завантаження даних додатку на сервер. Next.js відкриває можливість здійснювати миттєві навігації між іншими сторінками Оркестратора. Як команди мігрують свої маршрути в Оркестратор, вони автоматично стають маршрутами односторінкового додатку, що отримують переваги миттєвої навігації між іншими маршрутами Оркестратора.

Впровадження Next.js як основи для Фронтенд Оркестратора в Personio є ключовим кроком у підвищенні ефективності та гнучкості розробки. Використання Next.js, що базується на React і підтримує TypeScript [5], забезпечує зручне та знайоме середовище для розробників, уніфікуючи підходи до розробки та знижуючи поріг входження. Створення єдиного контексту рендерингу за допомогою Next.js і React сприяє більшій консистентності взаємодії користувачів із додатками, а також зменшує потребу в багаторазовій конфігурації та налаштуванні бібліотек для кожного окремого додатку. Це

спрощує процес інтеграції та розвитку фронтенд-додатків у рамках загальної екосистеми Personio. Використання власного сервера Express для Next.js дозволяє здійснювати важливі бізнес-критичні функції, забезпечуючи високий рівень контролю над процесом обробки запитів. Серверна обробка рендерингу, що підтримується Next.js, подальше покращує досвід завантаження сторінок для кінцевих користувачів, забезпечуючи швидкий та ефективний доступ до вмісту. Нарешті, впровадження Next.js як односторінкового додатку у фреймворку Оркестратора відкриває можливості для покращення навігації та взаємодії з додатками. Це дозволяє командам легко інтегрувати свої маршрути в Оркестратор і використовувати переваги миттєвої навігації, підвищуючи загальну продуктивність та задоволеність користувачів.

Таким чином, використання Next.js в Personio є стратегічним вибором, що сприяє технологічному зростанню та підвищенню ефективності фронтенд-платформи.

3.4.1 Маршрутизація Файлової Системи у Фронтенд Оркестраторі на базі Next.js

Фронтенд Оркестратор в Personio, будучи додатком на базі Next.js, активно використовує Маршрутизацію Файлової Системи Next.js. Це означає, що всі маршрути у Фронтенд Оркестраторі представлені у вигляді фізичних файлів у джерелі коду Оркестратора. Коли команди інтегруються в Фронтенд Оркестратор, їм необхідно усвідомлювати, як створювати шляхи в Next.js та як найкраще використовувати маршрутизатор Next.js, обираючи оптимальну структуру шляхів, яка задовольнятиме їхні потреби. Ідеально, коли команди створюють більш специфічні маршрути, уникаючи "всемогутніх маршрутів", які контролюють цілі підшляхи та ускладнюють розуміння обробки маршрутів. Статичні маршрути є найзручнішим способом створення маршруту в Фронтенд

Оркестраторі. Вони прості, прямолінійні та декларативні, що дозволяє точно визначити, на якому URL вони та відповідні їм фронтенд-додатки працюють.

Динамічні маршрути використовуються, коли маршрут містить змінну, яка потребує доступу в режимі реального часу. За допомогою динамічних маршрутів можна отримати доступ до значення маршруту в режимі реального часу через useRouter. Динамічні маршрути можуть використовуватися у будь-якому місці маршруту з тією ж поведінкою. Однак використання маршрутів захоплення та опціональних маршрутів захоплення ми рекомендуємо уникати, де це можливо, через їх здатність створювати "всемогутні маршрути". Такі маршрути мають широкий контроль над усіма підшляхами на певному маршруті та ускладнюють розуміння того, які шляхи обробляються Оркестратором.

/apps/orchestrator/pages/my-page.tsx (personio.de/my-page)

```tsx
import React from "react";
import { Integration as MyApp } from "@personio-web/my-app-integration";
import PageTitle from "../../../components/PageTitle/PageTitle";

const MyAppPage = () => (
  <>
    <PageTitle translationKey="my-app-title" />
    <MyApp />
  </>
);

export default MyAppPage;
```

Рисунок — 22 приклад статичної маршрутизації

/apps/orchestrator/pages/profile/[id].tsx (personio.de/profile/1337)

```tsx
import React from "react";
import { useRouter } from "next/router";
import { Integration as ProfileApp } from "@personio-web/profile-integration";
import PageTitle from "../../../components/PageTitle/PageTitle";

const ProfileAppPage = () => {
  const router = useRouter();
  return (
    <>
      <PageTitle translationKey="profile-app-title" />
      <ProfileApp userId={router.query.id} />
    </>
  );
};

export default ProfileAppPage;
```

Dynamic Routes aren't limited to the page-level file and can be used *anywhere* in the route, with the same behaviour:

/apps/orchestrator/pages/[id]/profile.tsx (personio.de/1337/profile)

```tsx
import React from "react";
import { useRouter } from "next/router";
import { Integration as ProfileApp } from "@personio-web/profile-integration";
import PageTitle from "../../../components/PageTitle/PageTitle";

const ProfileAppPage = () => {
  const router = useRouter();
  return (
    <>
      <PageTitle translationKey="profile-app-title" />
      <ProfileApp userId={router.query.id} />
    </>
  );
};

export default ProfileAppPage;
```

Рисунок — 23 приклад динамічної маршрутизації

3.4.2 Макети Сторінок у Фронтенд Оркестраторі

Фронтенд Оркестратор відповідає за загальний макет і структуру сторінок, забезпечуючи те, що сторінки по всьому Personio дотримуються послідовного вигляду і відчуття, знайомого користувачу. Стандартний макет містить усі елементи та структури для створення типової сторінки Personio. Він включає такі елементи, як навігація, верхня панель та пошук. Основна область контенту, де буде розміщений ваш додаток, є гнучким елементом, який автоматично розширюється до 100vh, дозволяючи вам повний контроль над макетом та вирівнюванням внутрішніх елементів вашого додатку. Оркестратор також пропонує додаткову опцію макету на повний екран, яку сторінки можуть

використовувати, якщо їх конкретний додаток вимагає використання всього відображуваного простору без додаткових елементів макету. Варто уточнити використання цього конкретного макету сторінки з командою UXP, оскільки він відрізняється від стандартної структури додатку. Важливо відзначити, що макет на повний екран видаляє деякі візуальні елементи, такі як навігація та верхня панель, але не видаляє базові функції, такі як можливість вживання імені іншої особи та сторонні скрипти. Приклад використання макету на повний екран у файлі /apps/orchestrator/pages/my-page.tsx демонструє, як можна встановити макет на повний екран для конкретної сторінки.

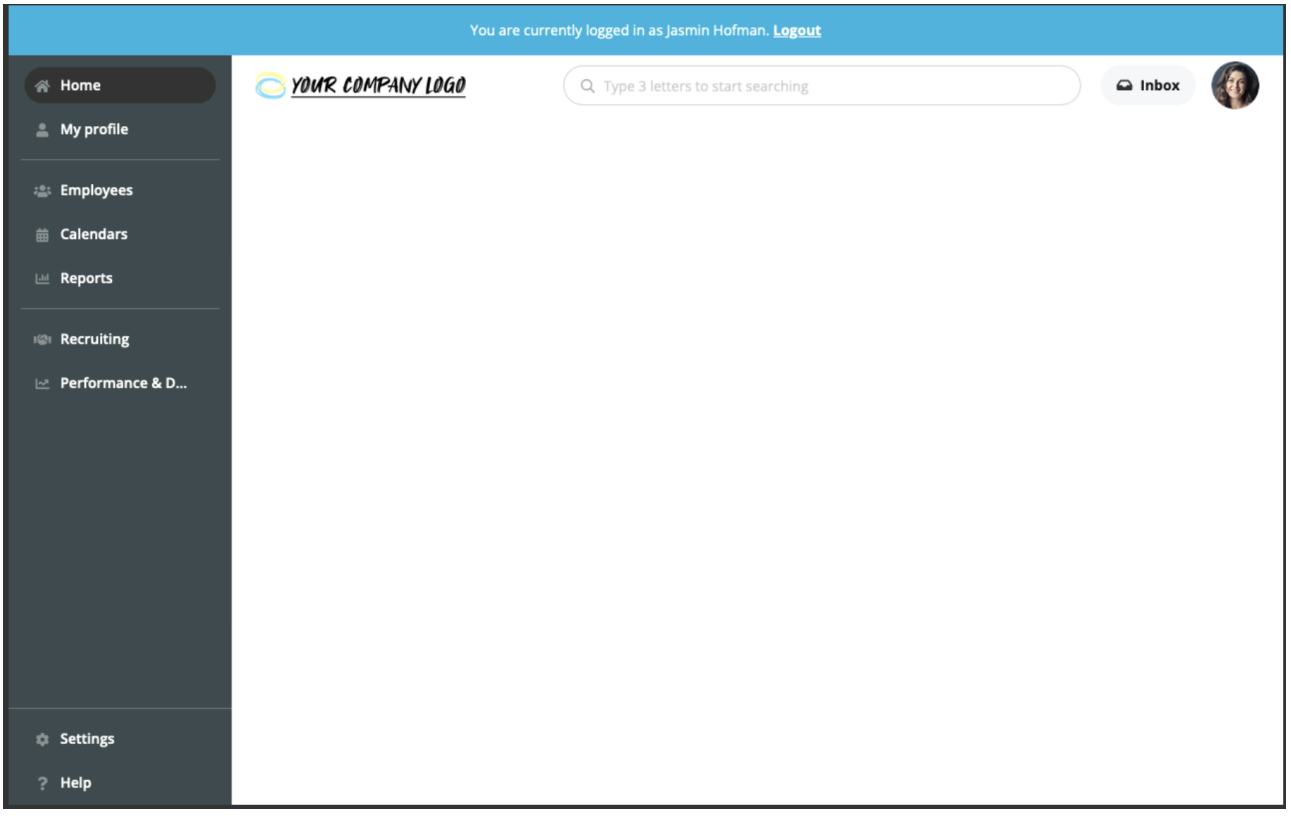

Рисунок — 24 Макет за замовчуванням

3.4.3 Вивчення універсальних провайдерів у фронтенд Оркестраторі

У світі сучасних веб-технологій, Фронтенд Оркестратор в Personio використовує низку універсальних провайдерів, кожен з яких відіграє свою унікальну роль у створенні координованого та ефективного користувацького

досвіду. Розглянемо кожну з цих технологій, щоб краще зрозуміти їх значення та функціонал.

Перший важливий елемент — це Split, який використовується як засіб управління функціональними флагами. Він дозволяє командам динамічно керувати доступом до певних функцій або компонентів додатку, що є ключовим для реалізації стратегій A/B тестування чи поступового впровадження нових функцій. Завдяки Split, команди можуть безпечно експериментувати та оптимізувати користувацький досвід, не наражаючи на ризик стабільність загальної системи. Далі іде React Query , що слугує для зручності збору даних. У Personio, React Query [17] використовується для управління запитами на сервер та керування кешуванням даних, що значно спрощує роботу з асинхронними даними та забезпечує плавне взаємодію з користувачем. Оркестратор надає глобальний QueryClientProvider, який охоплює всю дерево додатків, використовуючи стандартну конфігурацію від React Query. Це забезпечує єдиний підхід до обробки даних у всіх фронтенд-додатках, що рендеряться оркестратором.

Інтернаціоналізація, хоч і не реалізована через окремий провайдер, також відіграє важливу роль. Фронтенд Оркестратор ініціалізує бібліотеку i18next з налаштуваннями, які дозволяють завантажувати простори перекладу за запитом. Це означає, що всі фронтенд-додатки можуть використовувати react-i18next та компоненти <Translate /> від Personio Web, забезпечуючи безпроблемну локалізацію та переклади, що є ключовими для глобального досягнення та задоволення потреб різномовних користувачів.

Нарешті, маємо Datadog RUM, який працює на всіх сторінках Оркестратора. Datadog забезпечує моніторинг продуктивності веб-додатків, дозволяючи командам аналізувати та відстежувати користувацькі сесії, виявляти проблеми продуктивності та поліпшувати загальну якість додатків. Фронтенд-

додатки можуть додавати власну інформацію до трасувань RUM через Глобальний Контекст або власні таймінги продуктивності.

Кожен з цих провайдерів сприяє створенню більш злагодженого, ефективного та гнучкого середовища розробки, дозволяючи командам Personio фокусуватися на інноваціях та поліпшенні користувацького досвіду.

3.4.4 Використання Split.IO та впровадження функціональних прапорців для контрольованного розгортання

У Personio ми активно використовуємо Split.io разом з нашими власними бібліотеками-обгортками та додатковими інструментами, щоб швидко та безпечно доставляти функції нашим клієнтам. Наша поточна політика використання дозволяє прямий доступ до виробництва будь-кому, хто подає запит на обліковий запис Split.io.Ми сподіваємося, що це мінімізує труднощі для розробників та максимізує використання функціональних прапорців для підходу "рухатися швидко, не руйнуючи речей". Ми хочемо зберегти цей легкий підхід, який надає командам можливість активувати свої функції у виробництві самостійно, дотримуючись філософії "ви створюєте це, ви керуєте цим". Однак, враховуючи можливість неправильного використання цієї платформи різними шляхами, ми просимо всі команди витратити час на розуміння та дотримання кращих практик використання функціональних прапорців, щоб зберегти цей спосіб роботи. Важливі Аспекти

Уникайте створення Функціональних Прапорців, які повинні бути реалізовані як Опція Конфігурації. Split.io не є заміною для вашої бази даних. Будь ласка, переконайтеся, що ваш функціональний прапорець використовується лише для контролю вашого випуску і має заплановану дату закінчення життя, після якої всі посилання у кодовій базі будуть видалені. Не залишайте функцію наполовину впровадженою. Ми розуміємо, що плани іноді змінюються, в цьому випадку, будь ласка, відмініть впровадження вашого функціонального прапорця і перемістіть його в статус "видалено з коду" на дошці впровадження. Ви можете

створити його знову пізніше, якщо це буде потрібно. Не дозволяйте йому перебувати кілька місяців у напівзавершеному стані.

Накопичення великої кількості довготривалих Функціональних Прапорців неминуче призводить до створення кількох довгострокових проблем, які знижують надійність та якість нашого продукту.

3.4.5 Створення функціональних прапорців

Використання Split.io для Розгортання Змін за Допомогою Функціонального Прапорцяю. Розгортання змін за допомогою функціонального прапорця в Split.io зазвичай включає наступні кроки: Створення Нового Функціонального Прапорця в Split.io: Після входу в Split Dashboard через ваш портал OneLogin та вибору опції навігації по прапорцям, вам буде представлена форма для створення. При іменуванні нового функціонального прапорця важливо дотримуватися конвенції: <ID задачі Jira>-<описове ім'я прапорця>, наприклад, DX-4583-ім'я-моєї-чудової-функції. Створення Нових Сегментів за Потреби: Це необхідно, якщо ви плануєте розгортати вашу функцію, цілячись на групу співробітників або компаній у сегменті, який ще не був створений у списку доступних сегментів на Split Dashboard. Початок Використання у Кодовій Базі: Тепер можна почати оцінювати новий прапорець з відповідної кодової бази. Важливо мати кожен функціональний прапорець, призначений для однієї кодової бази, щоб уникнути складних для відлагодження проблем у виробництві. Розгортання Змін у Виробництво: Після того, як ви впевнені у поведінці вашого функціонального прапорця, ви можете розпочати його розгортання у виробництво. Видалення з Кодової Бази та Видалення з Split Dashboard: Після завершення розгортання вашої функції на 100% важливо оновити статус до "Видалено з Коду" після видалення всіх посилань у вашій кодовій базі. Тип Трафіку: Визначте, чи хочете ви розгортати зміни для однієї компанії за раз або для одного співробітника за раз. У більшості випадків у Personio ми

використовуємо тип трафіку компанії, оскільки зазвичай хочемо, щоб усі співробітники однієї компанії бачили однакову поведінку будь-якої конкретної функції. Теги: Введіть назву вашого племені та команди великими літерами для полегшення пошуку та фільтрації функціональних прапорців у інтерфейсі Split Dashboard.

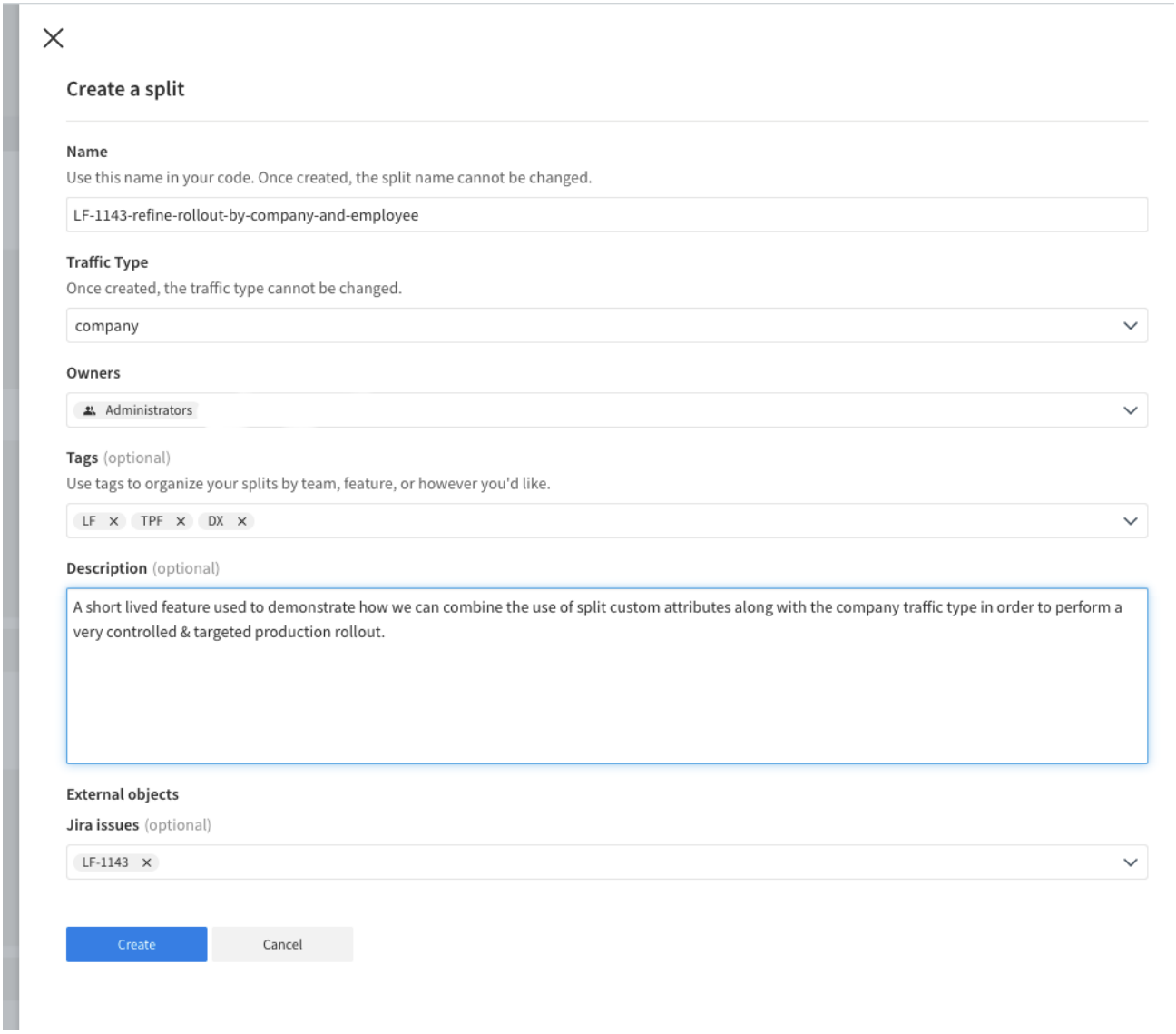

Рисунок — 25 Форма створенн функціонального прапорця

3.4.6 контрольоване розгоратння мікрофронтенда

Для покращення часу виведення змін у виробництво ми в процесі відмови від стадії тестування та рухаємося до безпечного тестування безпосередньо у

виробництві. Це вимагає від нас більш активного використання функціональних прапорців для безпечного випуску змін без необхідності виконання всебічних кінцевих тестів. Щоб безпечно тестувати у виробництві, мінімізуючи будь-який потенційний радіус поширення, ми можемо складати правила цільового використання, дотримуючись нижчеописаних кроків.

Загальна ідея полягає в тому, щоб спочатку активувати вашу функцію тільки для вас, а також для інших членів вашої команди, а потім тільки для співробітників Personio, після чого ми повинні достатньо протестувати функцію, щоб почати включати реальних клієнтів у розгортання. Для досягнення цього вимагається використання як значень ID компанії, так і ID співробітника для отримання додаткової точності у ваших правилах цільового використання. Ось приклад серії кроків, які дають нам цей рівень точності: Створіть правило цільового використання "активувати тільки для мене". Увійдіть до Personio у виробництві та отримайте свій ID співробітника з URL вашого профілю. Активуйте функцію тільки для себе, додавши наступне правило, яке використовує спеціальний атрибут ID співробітника (додаткові ID співробітників вашої команди можна просто додати до списку). Після початкового тестування ви можете безпечно збільшити вплив, не включаючи реальних клієнтів, додавши всіх співробітників Personio до розгортання, використовуючи розділ "Індивідуальні Цілі". Зверніть увагу, що правила цільового використання оцінюються в порядку пріоритетності, як показано в інтерфейсі користувача. Перша умова, яка відповідає, призупиняє процес і повертає результат. Наступні правила будуть пропущені. Почніть включати реальні компанійські акаунти, активуючи для 5% запитів, наприклад. З цього моменту ви можете вибирати більші чи менші інкременти до 100% випуску залежно від вашого судження.

Рисунок — 26 Вікно покрокового розгортання

Таким чином смислення використання Оркестратора, Split.io та універсальних провайдерів у Personio відкриває перед нами образ сучасної, гнучкої та високоефективної системи розробки. Оркестратор, як центральний елемент архітектури мікрофронтендів, забезпечує єдиний контекст рендерингу, покращуючи взаємодію між окремими додатками та забезпечуючи консистентність користувацького інтерфейсу. Його роль у структуруванні загального макету сторінки, надання спільних провайдерів та конфігурацій створює уніфіковане середовище для всіх фронтенд-додатків. Split.io, з іншого боку, відіграє ключову роль у управлінні випуском функцій, дозволяючи Personio ефективно та безпечно впроваджувати нові зміни. Використання функціональних прапорців забезпечує гнучкість та контроль над розгортанням функцій, дозволяючи поступово та стратегічно впроваджувати інновації. Універсальні провайдери, такі як Split для управління функціональними прапорцями, React Query [8] для оптимізації запитів даних, [14] Datadog для моніторингу та інші, є фундаментальними елементами, які сприяють створенню злагодженої, продуктивної та безпечної екосистеми розробки. Ці інструменти не тільки спрощують повсякденну роботу розробників, але й забезпечують високу якість та надійність кінцевого продукту. Загалом, комбінація Оркестратора, Split.io та універсальних провайдерів у Personio демонструє передовий підхід до управління складними веб-додатками, який забезпечує ефективність, гнучкість та безпеку у процесі розробки та впровадження нових функцій.

3.5 Практиче використання мікрофронтендів на прикладі розробки додатку Attendance Time Clock

З прийняттям законодавства ЄС про облік робочого часу в 2019 році, що вимагає від європейських компаній створити об'єктивну, надійну та доступну систему для вимірювання тривалості робочого часу кожного працівника, виникла необхідність у розробці простого та зручного рішення для співробітників. Згідно з відгуками клієнтів та результатами опитувань, існуючі рішення в Personio вважаються складними та неефективними у використанні, що негативно впливає на задоволеність клієнтів.

Розробка додатку Attendance Time Clock є пріоритетною через його значний вплив на місячний повторюваний дохід (MRR), задоволеність клієнтів і репутацію Personio. Ця функція знаходиться серед чотирьох найбільш запитуваних, маючи велику фінансову вагу. Низький рейтинг NPS в області відвідування підкреслює необхідність поліпшення цієї функціональності. Основною цільовою аудиторією є HR-менеджери, які прагнуть впровадити просте у використанні рішення для обліку робочого часу, та співробітники, які бажають мінімізувати час, витрачений на реєстрацію своїх робочих годин. Досягнення цих цілей базується на попередніх дослідженнях та користувацьких тестуваннях додатку Time Clock, результати яких демонструють високу важливість і потенційну ефективність цього рішення. Вважається, що впровадження додатку Attendance Time Clock підвищить задоволеність співробітників, зменшить MRR, пов'язаний з незадоволеними угодами через прогалини у функціональності обліку часу, та змінить підход співробітників до обліку робочого часу, зокрема, стимулюватиме їх до більш об'єктивного його реєстрування. В контексті розробки Attendance Time Clock, мікрофронтенди в Personio використовуються для створення модульного, надійного та легко інтегрованого рішення. Це підход, який дозволяє командам зосередитися на специфіці окремих компонентів системи, що сприяє швидкому розвитку,

гнучкості та високій адаптивності до змінних потреб клієнтів та законодавчих вимог.

Розробка Attendance Time Clock як частини ширшої стратегії використання мікрофронтендів у Personio демонструє глибоке розуміння потреб ринку та здатність компанії швидко адаптуватися до цих вимог. Це не лише поліпшує користувацький досвід та задоволеність клієнтів, але й сприяє підвищенню ефективності та інноваційності в розробці продуктів.

3.5.1 Генерація та запуск мікрофронтенда

Процес створення мікрофронтенда "Attendance Time Clock" як фрагмента сторінки розпочинається з генерації через інтерфейс Nx, де ключові параметри виливаються у фундамент нашого додатку. Підготовка полягає у введенні основних даних: назви, сфери діяльності, відповідальних осіб та локалізаційного контексту. Зануримося у цей процес із зосередженістю та увагою до важливих нюансів, що формують кістяк мікрофронтенда."Attendance Time Clock" отримує ідентифікацію як фрагмент сторінки, що вказує на його модульну роль в більшому інтерфейсі. Опис додатку надає чітке уявлення про його призначення, забезпечуючи підставу для дальшої розробки та інтеграції. Сервіс, до якого належить цей компонент, позначається для систематизації та координації в рамках загальної архітектури Personio.

Слід встановити канал зв'язку у Slack для підтримки та обговорення мікрофронтенда, а також зазначити відповідальну команду для ясності розподілу обов'язків і ефективного управління проектом. На цьому етапі розглядається стадія розвитку сервісу і його значущість для основного процесу роботи компанії. Це дозволяє оцінити потенційні ризики і необхідність забезпечення стабільності та доступності.

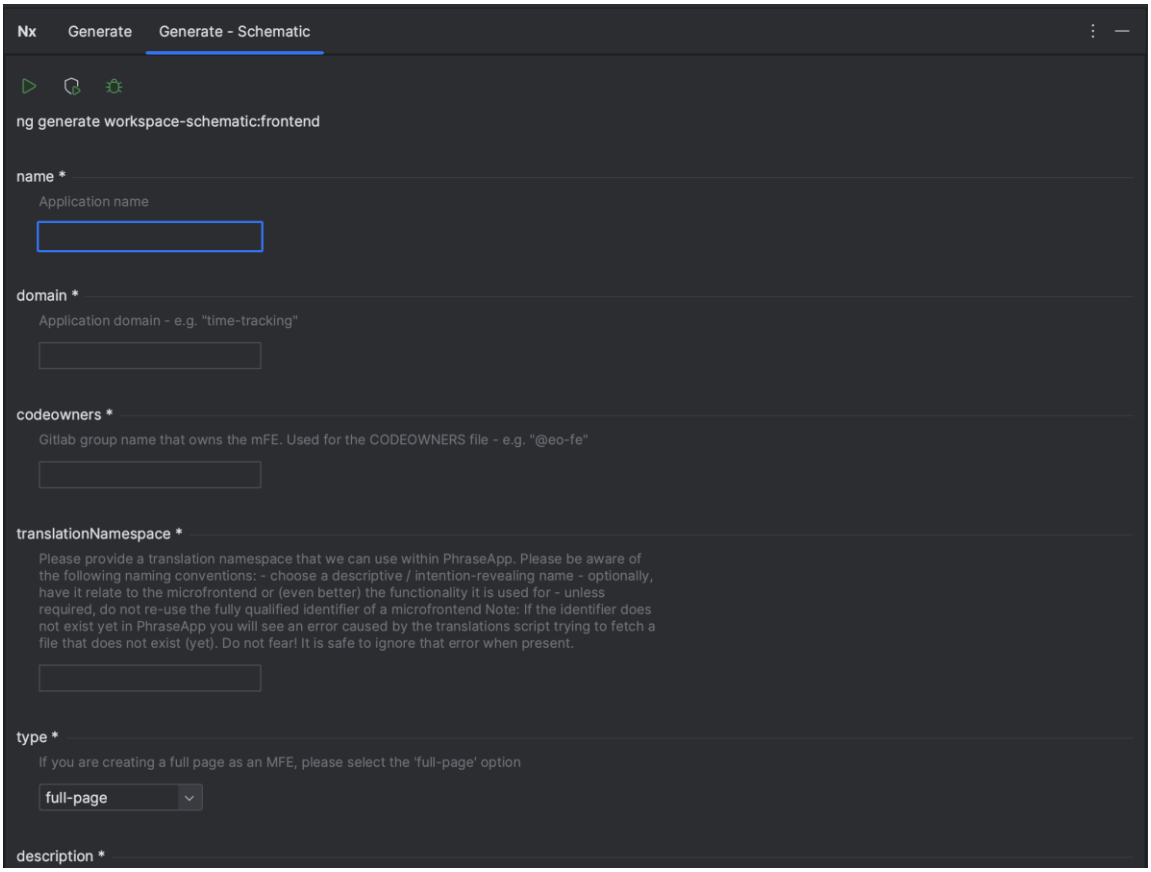

Рисунок — 27 Генератор мікрофронтенда з налаштуванням

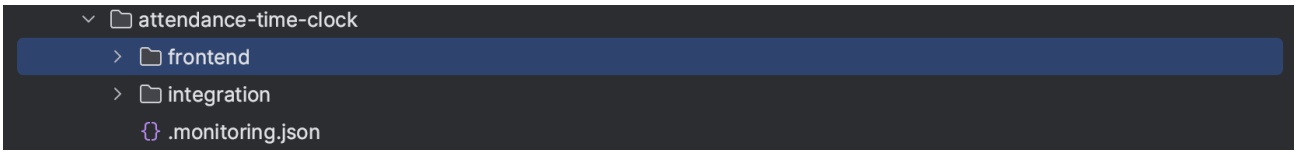

Рисунок — 28 Згенерована структура мікрофронтенда

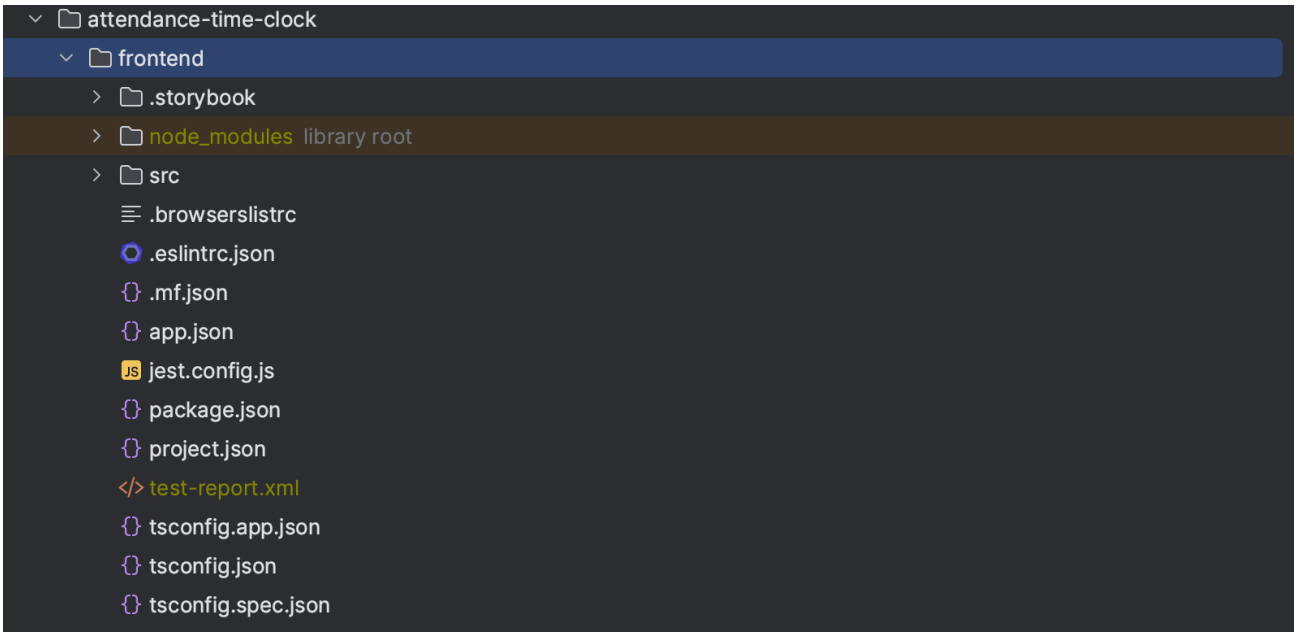

Рисунок — 29 Згенерована структура папки frontend

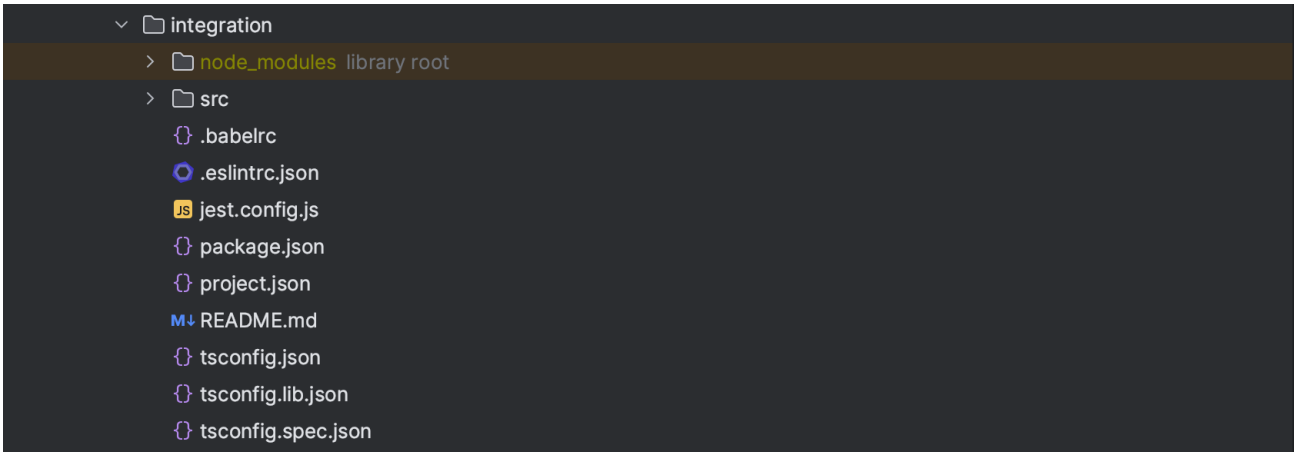

Рисунок — 30 Згенерована структура папки integration

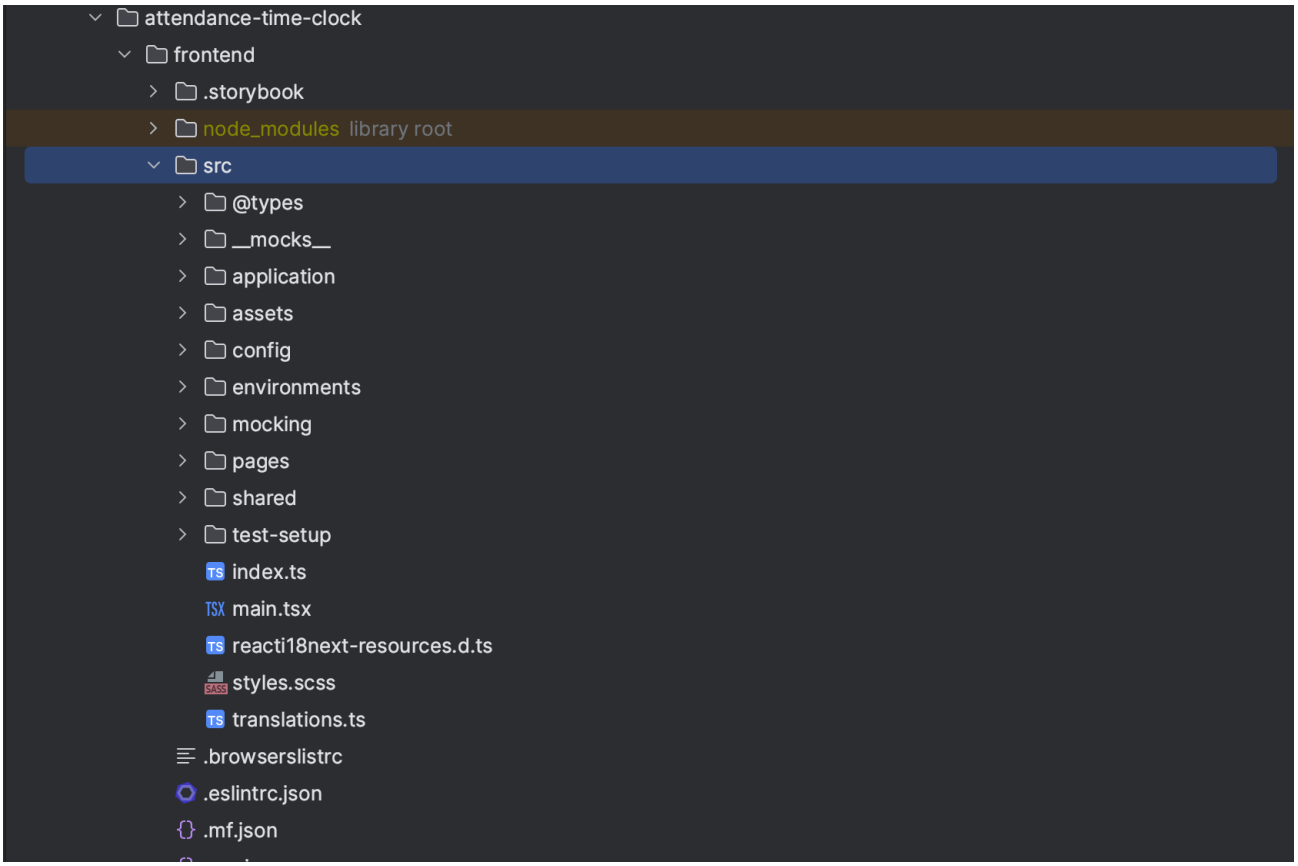

Рисунок — 31 Згенерована структура папки src

```
import ...

configureRUM();

configureRequest( {baseURL, timeout, retrieveDataFromResponse, dispatchSessionExpiredEvent: shouldDispatchSessionExpiredEvent, sessionExpiredD
  timeout: REQUEST_TIMEOUT,
  baseURL: API_BASE_URL,
});

const mfeConfig: MfeConfig = {
  mfeName: MICRO_FRONTEND_TRANSLATION_NAMESPACE,
  I18NOptions,
  mountingElement: MICRO_FRONTEND_ROOT_ELEMENT,
  renderWithSplitProvider,
};

const Wrapper: React.FC = () => {
  const isSidebarCollapsed : boolean  =
    localStorage.getItem( key: 'gx:isSidebarCollapsed') === '1';

  return <AppRoot isSidebarCollapsed={isSidebarCollapsed} />;
};

mocking.finally( onfinally: () => initMfe(Wrapper, mfeConfig));
```

Рисунок — 32 Згенерована структура файла main.tsx

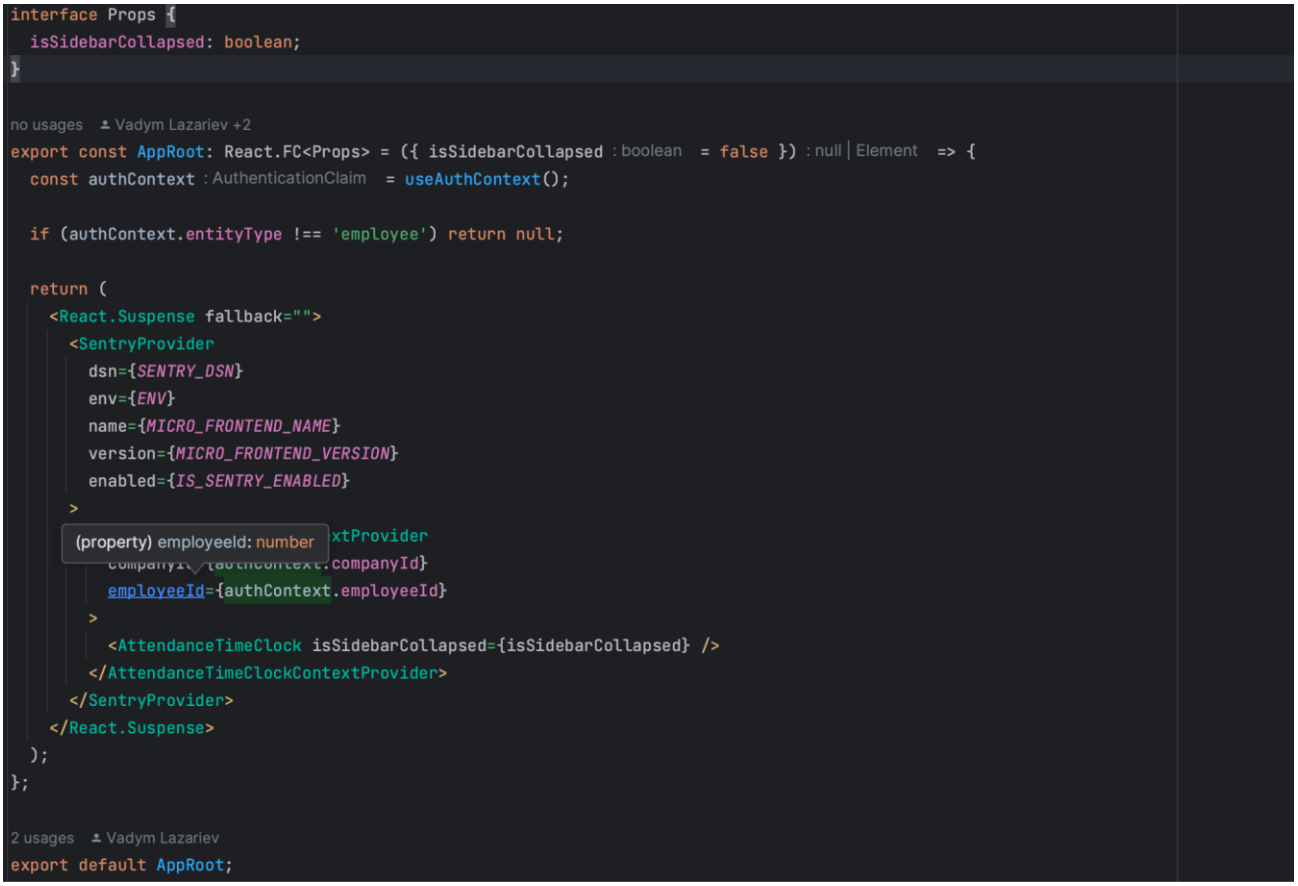

Рисунок — 33 Згенерована структура файла AppRoot.tsx

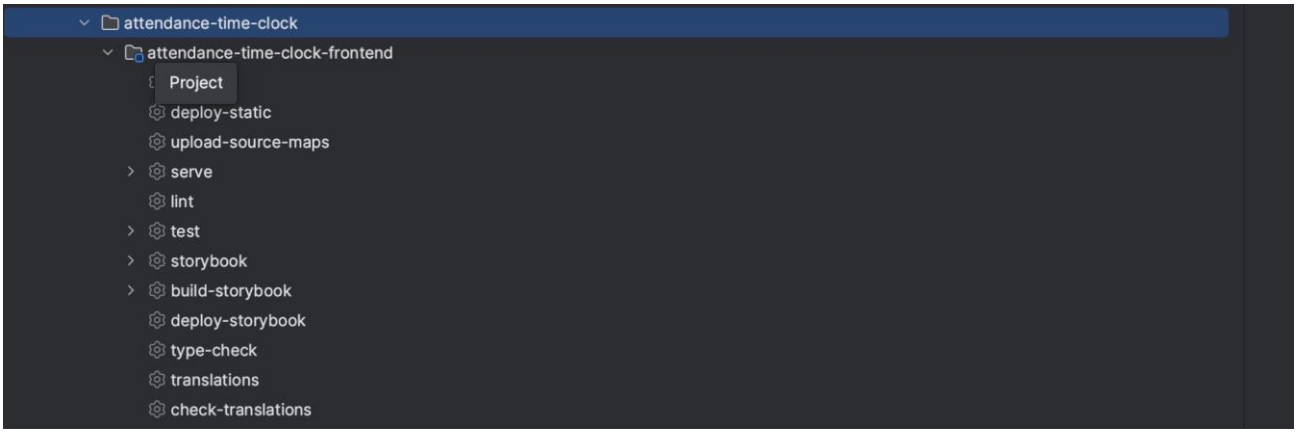

Рисунок — 34 Згенерована структура команд додатку

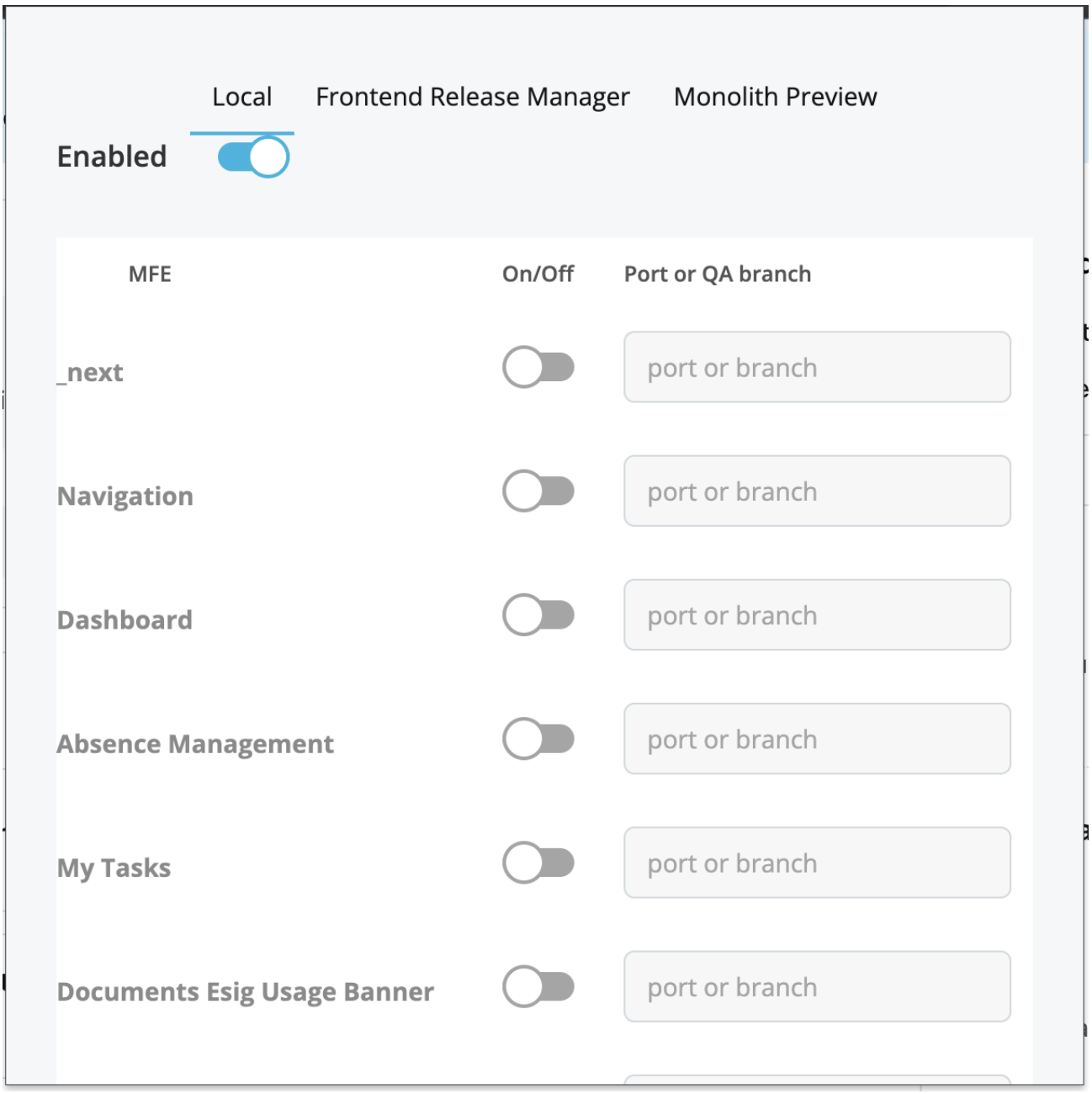

Рисунок — 35 Додаток Chrome для локального тестування мікрофронтендів

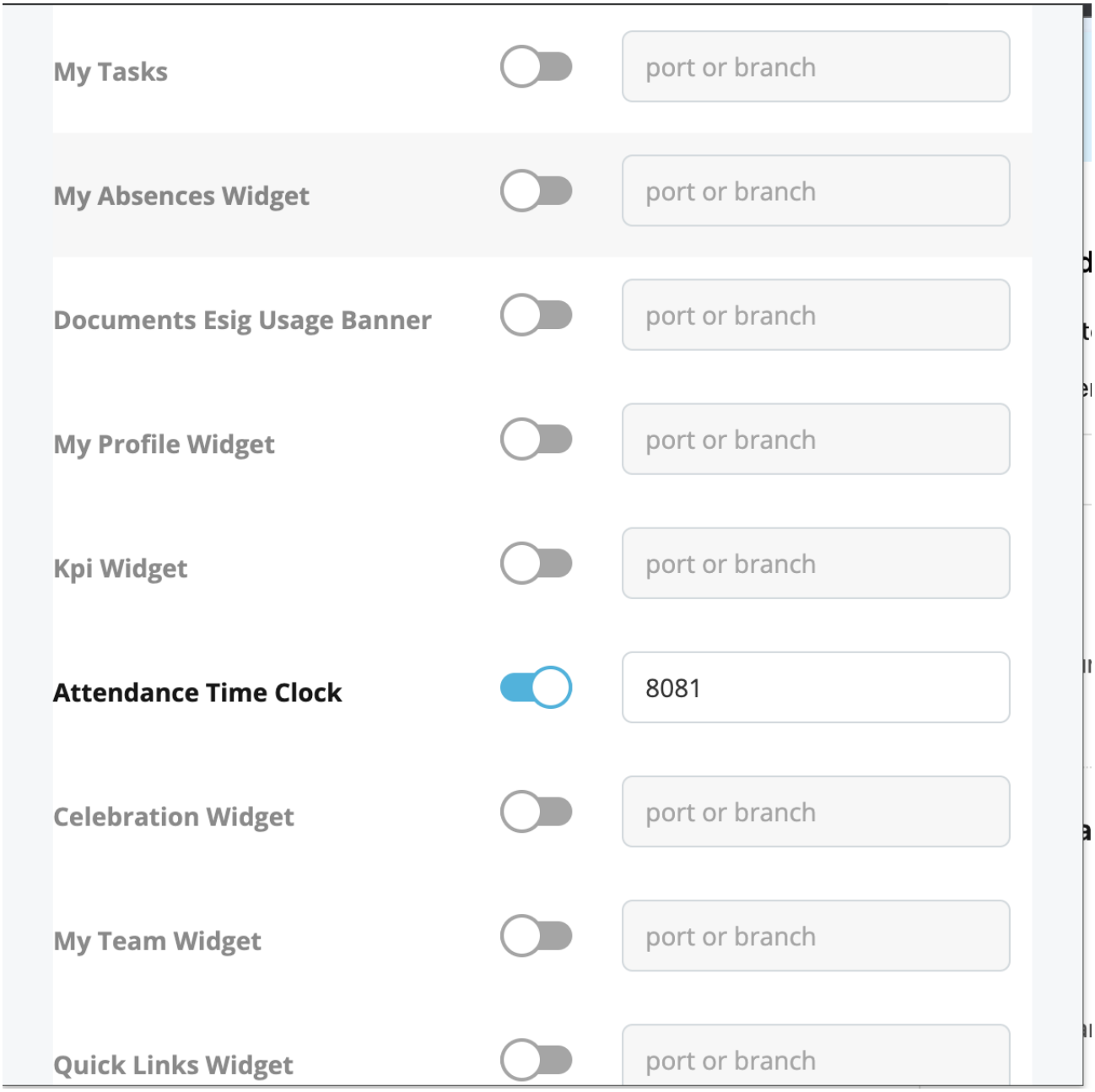

Рисунок — 36 Додаток Chrome з увімкненним мікрфронтендом

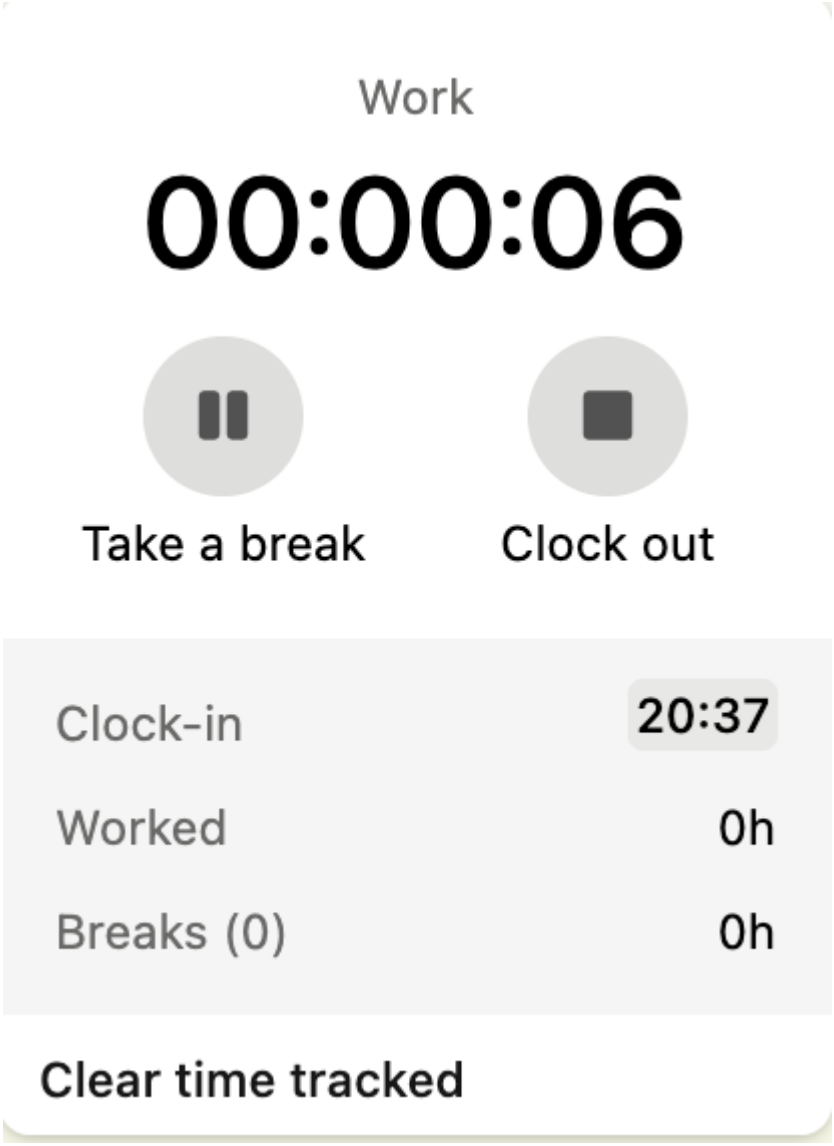

Рисунок — 37 Активний мікрофронтенд

3.5.2 Тестування мікрофронтенда

У процесі розробки "Attendance Time Clock", критично важливим є впровадження юніт-тестування. Ця практика дозволяє забезпечити високу якість коду та знизити ймовірність помилок, які можуть вплинути на користувацький досвід. Юніт-тести є невід'ємною частиною процесу безперервної інтеграції та доставки, допомагаючи швидко ідентифікувати та виправляти помилки на ранніх стадіях розробки.

Для написання та виконання юніт-тестів обрано Jest — потужний інструмент, який відзначається швидкістю виконання тестів та легкістю у конфігурації. Jest надає масивний набір утиліт для мокування, асертацій та паралельного виконання тестів, що робить його ідеальним вибором для масштабованих проектів. React Testing Library, у свою чергу, використовується для створення тестів, які більш зосереджені на поведінці компонентів, ніж на їх внутрішній стан. Цей інструмент стимулює розробників писати тести, керовані користувацьким досвідом, тобто тести, які імітують реальні взаємодії користувачів із додатком.

Юніт-тести розробляються з метою перевірки індивідуальних функцій та компонентів. Вони виконуються автоматично під час кожної збірки проекту, що гарантує, що будь-які зміни в коді, які можуть вплинути на існуючу функціональність, будуть відразу помічені. Юніт-тестування, здійснене з допомогою Jest та React Testing Library, є основою для створення міцного коду, що лягає в основу надійного мікрофронтенда Ретельно сплановані та виконані юніт-тести є гарантією якості, підвищуючи впевненість у тому, що новий функціонал не тільки задовольняє вимогам, але й не порушує вже існуючу логіку. Цей шар тестування дозволяє команді швидше впроваджувати зміни та ітерувати, зберігаючи при цьому довіру до стабільності продукту.

Юніт-тестування "Attendance Time Clock" є невід'ємною частиною розробки, що вимагає глибокого розуміння як бізнес-логіки, так і технічної сторони проекту. Використання Jest та React Testing Library дозволяє створити надійну основу для майбутньої розширюваності та підтримки, вкладаючи фундамент під здорову та ефективну екосистему мікрофронтендів у Personio.

```
describe('ActiveTimeClock', () : void => {
  describe('when user is working', () : void => {
    it('should render correctly when user is working and sidebar is not collapsed', () : void => {
      advanceTo(
        add(
          new Date(
            activeSessionWorkingNoPriorPeriods.data.current_period_start_date_time.value,
          ),
          duration: { minutes: 35, seconds: 10 },
        ),
      );
      renderActiveTimeClock(
        <ActiveTimeClock
          isSidebarCollapsed={false}
          timeClockStatus={AttendanceTimeClockStatus.WORKING}
          timeClockActiveSession={timeClockActiveSessionWorkingNoPriorPeriods}
        />,
      );

      expect(getTakeABreakButton()).toBeInTheDocument();
      expect(getSeeMoreOptionsButton()).toBeInTheDocument();
```

Рисунок — 38 Приклад юніт-теста

3.5.3 Розгортання мікрофронтенда

Процес розгортання мікрофронтенда "Attendance Time Clock" вимагає ретельно налаштованого пайплайну, який гарантує автоматизоване та безперебійне доставлення оновлень продукту до кінцевих користувачів. Це включає в себе комплекс дій та процедур, визначених у YAML-файлах, що забезпечують гнучкість та ефективність процесу розгортання. Основний файл конфігурації пайплайну розташований в основній директорії репозиторію, він інтегрує всі індивідуальні кроки, сценарії та утиліти, створюючи єдине середовище для GitLab.

Додаткові файли конфігурації для окремих кроків та утиліт знаходяться в директорії .gitlab, де кожен з них має свою відповідальність і призначення. Динамічні завдання пайплайну створюються на льоту та представляють собою дочірні процеси, які генеруються під час роботи завдання create-dynamic-jobs.

Використовуючи скрипти на JavaScript, створюються YAML-файли, які описують дії дочірніх пайплайнів, тим самим забезпечуючи гнучкість та адаптивність процесу розгортання. Скрипт createJobs.js використовується для визначення змінених проектів і відповідних цілей, які необхідно запустити. За допомогою цього скрипта формуються конфігурації завдань, які потім перетворюються у формат YAML та зберігаються у директорії dynamic_jobs як артефакти пайплайну.

В директорії jobs є файл dynamic-jobs.yml, який визначає батьківські завдання для запуску дочірніх пайплайнів і відповідних завдань у них. Ці завдання використовують артефакти, згенеровані в create-dynamic-jobs, та активують відповідні процеси в GitLab для реалізації завдань. Конфігурація пайплайну для "Attendance Time Clock" є ключовим елементом для забезпечення послідовності та ефективності процесів розробки та розгортання.

3.5.4 Менеджер Релізів Фронтенду

Менеджер Релізів Фронтенду (Frontend Release Manager, FRM) репрезентує суттєве удосконалення процесу розгортання та випуску мікрофронтендів, що входять до монорепозиторію Personio Web. Головні завдання FRM полягають у розділенні понять розгортання та випуску, веденні історії та статусів розгортань та релізів, наданні засобів для швидкого випуску, відкату та позначення розгортань, а також візуалізації поточного та історичного стану релізів для простого оцінювання та взаємодії.

Проект націлений на забезпечення більшого контролю над версіями мікрофронтендів, які ми надаємо нашим клієнтам, моніторинг стану та впливу релізів і швидкий відкат помилкових розгортань для зменшення середнього часу відновлення сервісу з сотень хвилин до декількох секунд.

Розгортання - це запис у FRM, статус якого неактивний. Це дозволяє розділити поняття розгортання та випуску продукту. Випуск - це запис у FRM,

який раніше був розгортанням, але тепер має статус активний або нестабільний. Активний статус - випуск, який може бути наданий клієнтам, якщо це останній реліз. Неактивний статус - визначає розгортання, що ще не були випущені - записи з цим статусом не можуть бути надані клієнтам. Нестабільний статус - визначає реліз, який раніше був активним та доступним для клієнтів, але міг викликати інцидент і тому був відкочений.

Менеджер Релізів Фронтенду (FRM) є стратегічним інструментом, який надає командам розробників Personio більшу гнучкість і контроль над процесом випуску їхніх продуктів. Він відіграє ключову роль у підвищенні якості користувацького досвіду, зменшуючи ризики, пов'язані з випуском нових функцій, та гарантуючи стабільність та надійність сервісів.

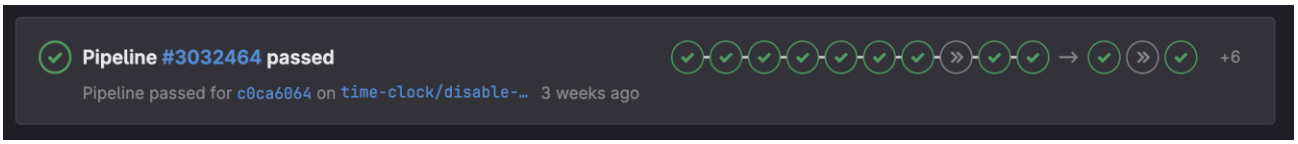

Рисунок — 39 Приклад успішного пайплайна

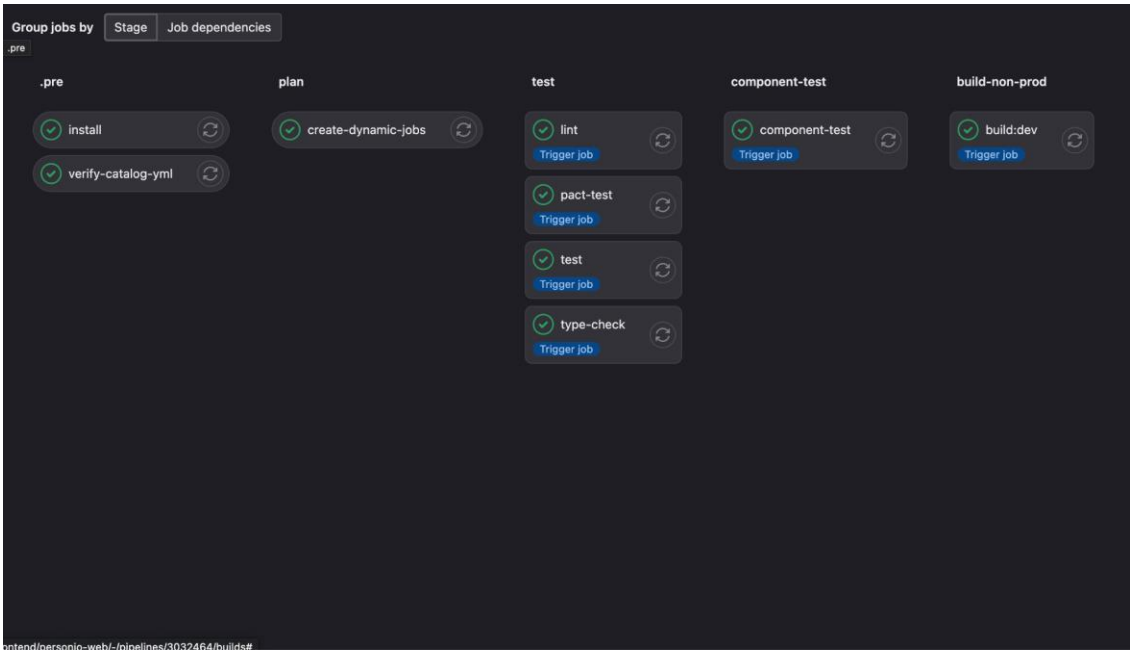

Рисунок — 40 Приклад праць які запускаються в пайплайні

3.5.5 Моніторінг та виявлення критичних помилок в production середовищі

У світі, де швидкість і надійність веб-застосунків можуть мати критичний вплив на успіх підприємства, забезпечення безперебійної роботи продуктивного середовища є важливим завданням для будь-якої команди розробників. [14] Datadog і синтетичні тести є двома потужними інструментами, що дозволяють не лише моніторити стан застосунків у реальному часі, але й прогнозувати та виявляти проблеми, перш ніж вони стануть відчутними для кінцевого користувача. Datadog пропонує комплексне рішення для моніторингу, яке дає можливість візуалізувати метрики, логи та трасування на єдиній платформі. Завдяки інтеграції Datadog у продуктивне середовище, можна не тільки слідкувати за загальною продуктивністю системи, але й детально аналізувати роботу окремих мікросервісів або мікрофронтендів.

Це стає незамінним при виявленні затримок у відповідях сервера, несподіваних збоїв або витоків пам'яті, що можуть призвести до падіння застосунку. Datadog дозволяє виявляти аномалії у поведінці застосунку, використовуючи алгоритми машинного навчання для аналізу історичних даних, що може бути особливо корисним для прогнозування майбутніх проблем. Також, сервіс надає інструменти для налаштування сповіщень, які миттєво інформують команду розробників про виникнення критичних подій, що потребують негайного втручання.

Синтетичні тести використовуються для імітації користувацьких дій та перевірки функціональності веб-застосунків. Ці тести дозволяють систематично перевіряти доступність і реактивність різних ендпойнтів і сервісів, що є частиною застосунку, тим самим забезпечуючи безперервний контроль якості. За допомогою синтетичних тестів можна виявити проблеми, які не були враховані під час ручного тестування або не відтворюються у стандартних сценаріях користувача. Ключовою перевагою синтетичних тестів є можливість їх

виконання у регулярному автоматичному режимі, що дозволяє підтримувати високий рівень готовності застосунку до роботи під високим навантаженням або в умовах нестандартних ситуацій. Такі тести можуть бути налаштовані на виконання з різних географічних локацій, що дозволяє оцінювати продуктивність застосунку з точки зору користувачів з різних регіонів.

Об'єднання можливостей [14] Datadog і синтетичних тестів створює потужну систему для забезпечення стабільності та продуктивності веб-застосунків. Синтетичні тести генерують додаткові дані про поведінку застосунку, які можуть бути інтегровані у Datadog для розширення аналітичних можливостей і вдосконалення системи сповіщень.Можливість швидкого реагування на проблеми, виявлені у ході синтетичного тестування, забезпечує високий рівень надійності застосунків, а також допомагає уникнути великих проблем, які можуть вплинути на досвід користувачів та фінансові показники компанії.

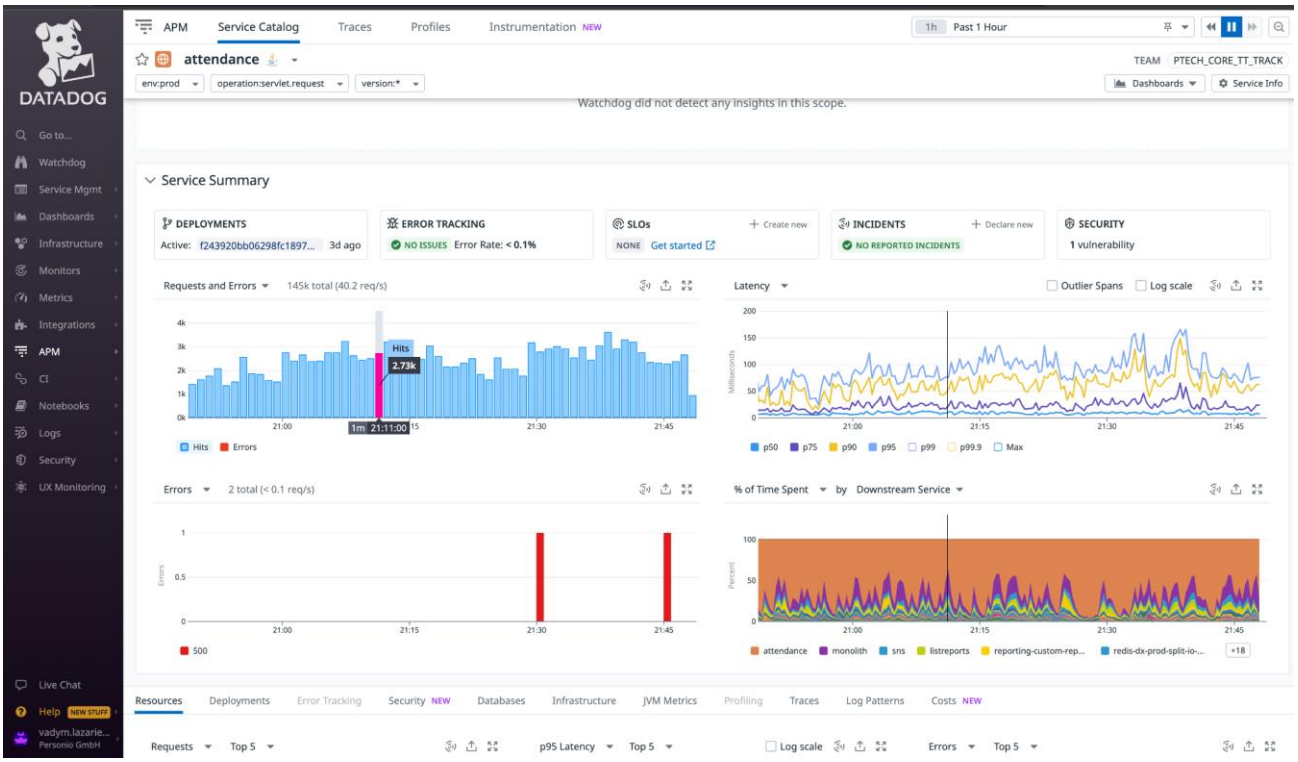

Рисунок — 41 Сторіка Datadog для моніторингу сервісів

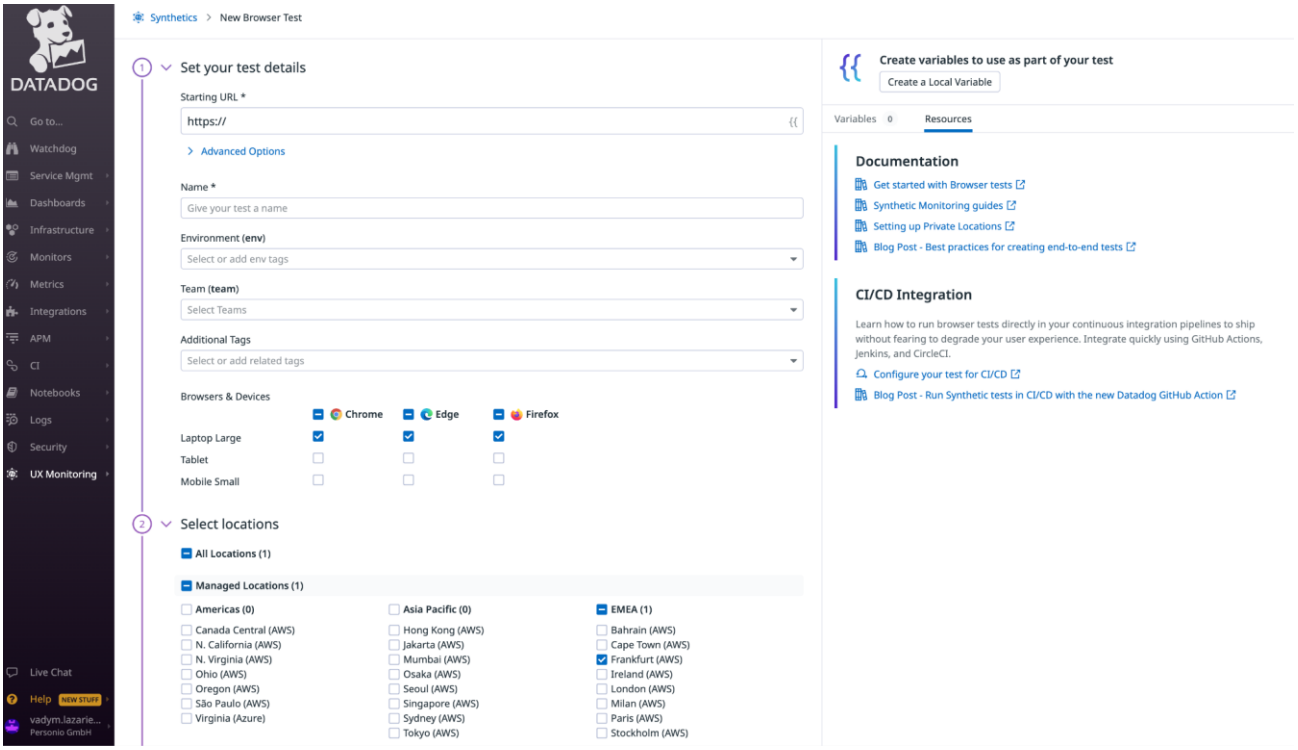

Рисунок — 42 Сторінка Datadog створення сінтетичних тестів

STATUS	TYPE	NAME	DOMAIN	TAGS	ENVS	TEAMS	UPTIME	LAST MODIFIED
Edit Tags	Edit Envs	Edit Teams	Run Tests Now	Delete				
OK	Browser	[HRM] Create an attendance e...		domain:corehr +3	prod	PTECH_CORE_T...	100%	4 Weeks Ago
OK	Browser	[HRM] Add work from attenda...		coreFlow:atte... +2	prod	PTECH_CORE_T...	100%	2 Weeks Ago
OK	Browser	[HRM] Attendance Dashboard...		coreFlow:Atte... +2	prod	PTECH_CORE_T...	100%	4 Weeks Ago
OK	Browser	[HRM] Modify attendance end...		coreFlow:upd... +2	prod	PTECH_CORE_T...	100%	4 Weeks Ago
OK	Browser	[HRM] Modify attendance star...		coreFlow:upd... +2	prod	PTECH_CORE_T...	100%	4 Weeks Ago
OK	Browser	[HRM] Delete an attendance ...		coreFlow:atte... +2	prod	PTECH_CORE_T...	100%	4 Weeks Ago
OK	Browser	[HRM] Modify attendance bre...		coreFlow:upd... +2	prod	PTECH_CORE_T...	100%	4 Weeks Ago
OK	Browser	[HRM] Update project on atte...		coreFlow:atte... +2	prod	PTECH_CORE_T...	100%	4 Weeks Ago
OK	Browser	[HRM] Delete break from atte...		coreFlow:atte... +2	prod	PTECH_CORE_T...	100%	4 Weeks Ago
OK	Browser	[HRM] Modify attendance co...		coreFlow:upd... +2	prod	PTECH_CORE_T...	100%	4 Weeks Ago
OK	Browser	[HRM] Attendance Page Moda...		coreFlow:upd... +2	prod	PTECH_CORE_T...	100%	4 Weeks Ago
OK	Browser	[ATM] Import attendance peri...		domain:corehr +2	prod	PTECH_COREH...	100%	4 Days Ago
OK	Browser	[HRM] Restrict time adding - d...		coreFlow:upd... +2	prod	PTECH_CORE_T...	100%	4 Weeks Ago
OK	Browser	[HRM] Restrict time adding - d...		coreFlow:upd... +2	prod	PTECH_CORE_T...	100%	2 Weeks Ago
OK	Browser	[HRM] Restrict time adding - d...		coreFlow:upd... +2	prod	PTECH_CORE_T...	100%	4 Weeks Ago
OK	Browser	[HRM] Restrict Time Adding - ...		coreFlow:atte... +2	prod	PTECH_CORE_T...	100%	4 Weeks Ago
OK	Browser	[HRM] Add confirmed attenda...		coreFlow:atte... +2	prod	PTECH_CORE_T...	100%	4 Weeks Ago
OK	Browser	[HRM][TRACK] Add 9am-5pm ...		coreFlow:upd... +2	prod	PTECH_CORE_T...	100%	2 Weeks Ago

Рисунок — 43 Сторінка Datadog активних сінтетичних тестів

3.6 Висновки до Розділу 3

1. Personio прагнуло досягти більшої модульності та незалежності різних частин своєї веб-платформи, що дозволило б кожній команді розвивати та масштабувати свої функціональні можливості незалежно від інших.

2. Необхідність в гнучкості розгортання та управління версіями продуктів привела до впровадження Frontend Release Manager, який надав можливість швидкого реагування на помилки та ефективного управління релізами.

3. Використання Orchestrator - це стратегічне рішення, яке дозволяє Personio створювати єдиний контекст рендерингу для всіх внутрішніх фронтенд застосунків, забезпечуючи високий рівень консистенції та спрощуючи інтеграцію мікросервісів.

4. Інтеграція універсальних провайдерів, таких як Split.io, відіграє важливу роль у гнучкому управлінні функціональними можливостями та експериментами в продукті, дозволяючи командам "перемикати" функції без прямих змін у кодовій базі.

5. Виклики з моніторингу та виявлення помилок в середовищі продакшн спонукали до активного використання Datadog та синтетичних тестів для забезпечення стабільності та відмінної роботи застосунків.

6. Personio зіткнулося з викликами, пов'язаними з недостатністю об'єктивності та легкості використання існуючих рішень для відстеження робочого часу, що стало ключовим чинником для розробки нового рішення "Attendance Time Clock".

7. Оркестратор Personio використовується як централізований шар управління, який координує завантаження та відображення різних

мікрофронтендів, що дозволяє забезпечити однорідний користувацький досвід та оптимізувати процеси розгортання та масштабування.

8. Вибір оркестратора заснований на його здатності інтегрувати мікросервіси в єдиний користувацький інтерфейс, тим самим знижуючи зусилля, необхідні для розвитку та підтримки окремих компонентів, та сприяючи більшій когерентності продукту.

9. Ініціативи Personio щодо оркестрації та мікрофронтендів відображають загальну тенденцію в індустрії до більш розподілених та модульних архітектур, які забезпечують велику масштабованість та гнучкість у відповідь на зміни на ринку та вимоги користувачів.

10. Ці висновки підкреслюють важливість продовження інновацій та адаптації до нових підходів у розробці високопродуктивних та надійних веб-застосунків, забезпечуючи при цьому відповідність до вимог законодавства та збереження високого рівня задоволеності користувачів.

РОЗДІЛ 4. АНАЛІЗ РЕЗУЛЬТАТІВ ВПРОВАДЖЕННЯ МІКРОФРОНТЕНДНОЇ АРХІТЕКТУРИ У ВИСОКОНАВАНТАЖЕНИХ ВЕБ-ЗАСТОСУНКАХ

У четвертому розділі ми поглиблено аналізуємо результати впровадження мікрофронтендної архітектури в контексті високонавантажених веб-застосунків. Засновуючи нас на досвіді та аналізі, наведеному у третьому розділі, ми розглядаємо конкретні приклади та висновки, які випливають з реальної практики застосування цієї архітектурної парадигми.

Мікрофронтенди, як засіб організації фронтенду, надають значні переваги у плані швидкості розробки, масштабування та управління технічним боргом. Цей підхід дозволяє клієнтам завантажувати лише ті частини веб-застосунку, які їм потрібні в даному контексті, зменшуючи обсяг завантажуваних ресурсів і поліпшуючи швидкість веб-сторінок. В цьому розділі ми прагнемо глибше вивчити, як саме мікрофронтендна архітектура впливає на ці аспекти, розглядаючи питання продуктивності, ефективності та здатності до масштабування в реальних умовах. Ми досліджуємо різні стратегії та методи оптимізації, що застосовуються при роботі з мікрофронтендами, а також аналізуємо виклики та можливі рішення, які виникають під час їх інтеграції та експлуатації.

Важливим аспектом нашого аналізу є вивчення впливу мікрофронтендів на загальну продуктивність та стабільність веб-застосунків. Враховуючи широкий спектр можливостей, які вони надають, ми прагнемо виявити, як ці переваги можуть бути максимально реалізовані в контексті високонавантажених веб-застосунків.

Цей аналіз дає нам змогу заглибитися в питання оптимального використання мікрофронтендної архітектури, розглядаючи реальні випадки та надаючи цінні інсайти для подальшого вдосконалення та розвитку таких систем.

4.1 Результати впровадження мікрофронтендної архітектури у веб-застосунках

У цьому розділі ми зосереджуємо увагу на аналізі практичних результатів впровадження мікрофронтендної архітектури у розробку високонавантажених веб-застосунків. Як зазначено у попередніх розділах, особливо у третьому, використання мікрофронтендів відкриває нові можливості для гнучкості, ефективності та масштабованості веб-застосунків.Мікрофронтенди, як свідчить досвід провідних технологічних компаній, дозволяють розробникам реалізовувати більш інноваційні та адаптивні рішення. Наприклад, компанія Personio, яка використовує мікрофронтендну архітектуру, показала здатність до швидкого масштабування та ефективного впровадження нововведень. Створення модульних та незалежних частин веб-застосунку забезпечує більшу гнучкість та спрощує процес розробки, тестування та впровадження нового функціоналу. Наприклад міграція з монолітної архітектури на мікрофронтенди домпомогла знизити розгортання застосунку з декількох годин до декількох хвилин. Основні переваги мікрофронтендної архітектури, які стали помітні внаслідок її впровадження, включають збільшення продуктивності та швидкості реакції веб-застосунків, що є критично важливим для високонавантажених систем. Це досягається завдяки можливості завантажувати лише необхідні частини застосунку, що знижує загальний обсяг завантажуваних даних та покращує час відгуку веб-інтерфейсу.

Впровадженні мікрофронтендів має свої нюанси, пов'язані з необхідністю забезпечення ефективної координації та інтеграції між різними частинами системи. Особливу увагу у цьому контексті потребує рендерінг мікрофронтендів

у браузері, де ключовим завданням є оптимізація завантаження, взаємодія з глобальним середовищем браузера, та управління залежностями між різними частинами системи.

Подальший аналіз роботи Personio у розділі 3 вказує на значні переваги, які можуть бути досягнуті за допомогою мікрофронтендної архітектури, зокрема, з точки зору гнучкості розгортання та управління версіями продуктів. Впровадження інструментів, таких як Frontend Release Manager, надає можливість швидко реагувати на помилки та ефективно управляти релізами, підвищуючи тим самим оперативність та гнучкість у відповіді на мінливі вимоги ринку та потреби користувачів. Інтеграція з Orchestrator, який діє як централізований шар управління, координуючи завантаження та відображення різних мікрофронтендів, є стратегічним кроком, який дозволяє забезпечити однорідний користувацький досвід та оптимізувати процеси розгортання та масштабування. Orchestrator є однією за найважилвіших технологій в контексті стратегічного значення розробки додатків. Зміна PHP двигуна для рендерінгу сторінок на Orchestrator допомогла збільшити продуктивність на 47% і завантаження DOM на 40% відостків в окремих випадках.

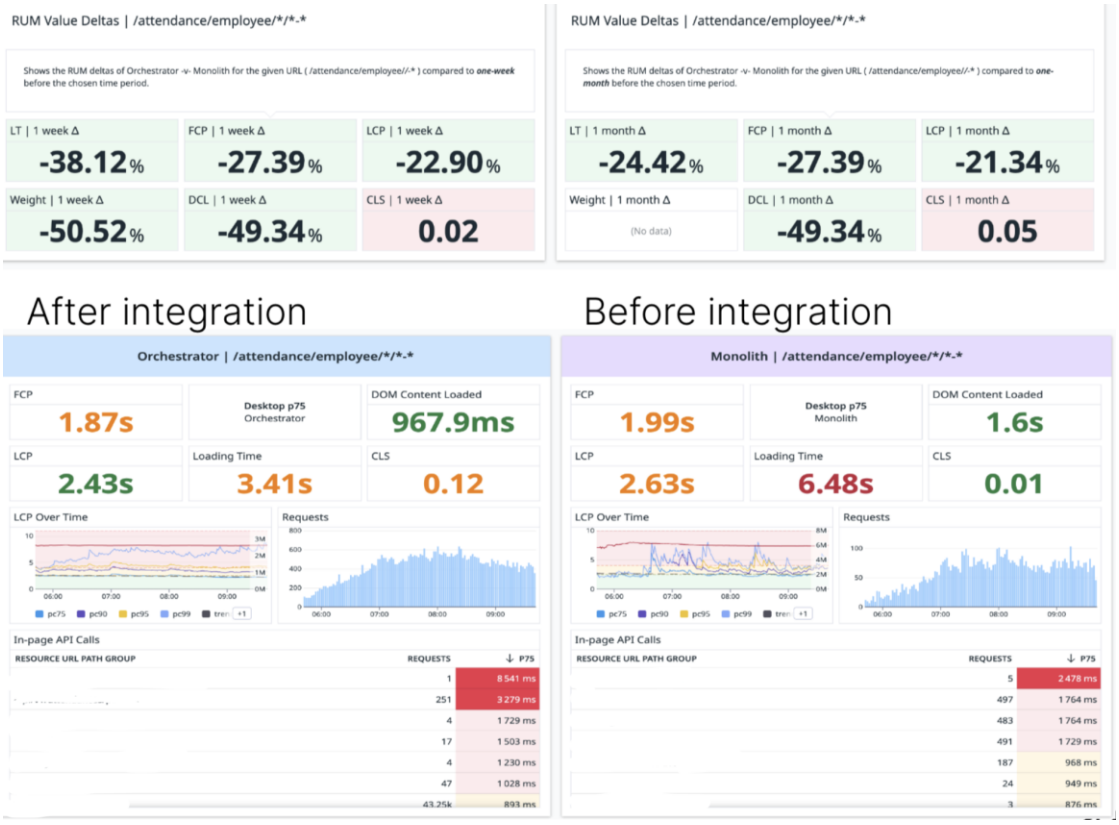

Рисунок — 44 Сторінка Datadog до/після інтеграції Orchestrator

Таким чином можна відмітити важливість продовження інновацій та адаптації до нових підходів у розробці високопродуктивних та надійних веб-застосунків, забезпечуючи при цьому відповідність до вимог законодавства та збереження високого рівня задоволеності користувачів. Саме такий підхід дозволяє відповідати на сучасні виклики в сфері веб-розробки, забезпечуючи надійність, ефективність та масштабованість веб-застосунків.

4.1.1 Аналіз продуктивності мікрофронтендних застосунків

Розглядаючи внутрішню механіку та динаміку мікрофронтендних застосунків, ми зіштовхуємося з важливістю усвідомлення того, як ці архітектурні рішення впливають на загальну продуктивність. Поглиблене вивчення метрик, таких як час завантаження сторінок, відкриває широкий простір для оцінки ефективності мікрофронтендів.

Використання [14] Datadog як інструменту для моніторингу продуктивності дозволяє нам зануритися у світ точних даних та глибокого аналізу. Особлива увага приділяється стабільності сінтетичних тестів, де ми виявляємо тенденції в роботі застосунків, а також графікам розгортань, які ілюструють динаміку розвитку та оновлення проекту. Інтеграція Datadog у мікрофронтенді є дуже важливим аспектом нагляду за стабільністю додатка, цей сервіс допомагає створювати ідивідуальні метрики які будуть використанні для окремого сервісу. Це допомогає нам створити статистику стабільності сервісів де в ретроспективі можна дослідити минулі помилки та запобігти їх в майбутньому. На основі цих метрик ми можемо порівняти завантаженність мікрофронтендних додатків або монолітних.

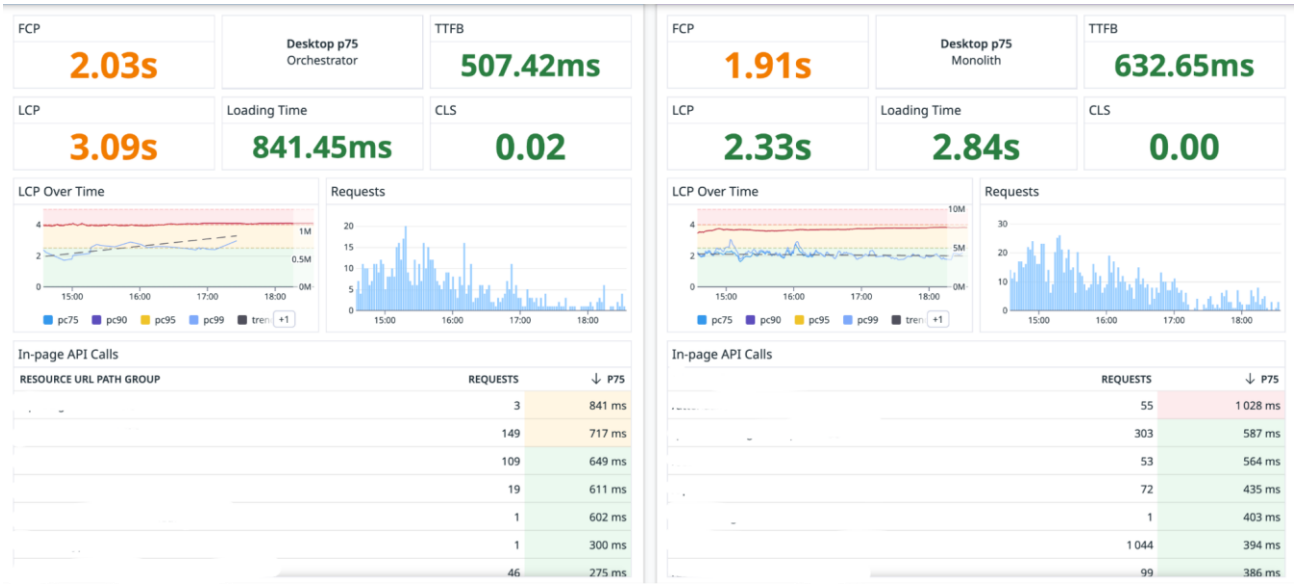

Рисунок — 45 Порівняння моноліту та мікрофронтенда

На даних метриках можна побачити час завантаження, затримку та швидкість рендерінгу. Основуючись на приведених даних в контексті цього застосунку монолітне рішення програє мікрофронтенду майже в два рази. Збільшення продуктивності безпосередньо впливає на покращення користувацького досвіду. Швидкість відгуку застосунку на запити користувача стає ключовим фактором у задоволенні їх потреб та очікувань. Це

підтверджується через позитивні відгуки користувачів, які відчувають покращення в використанні застосунків.

Аналізуючи мікрофронтенди у контексті порівняння з монолітними структурами, ми виявляємо значні переваги, особливо у площині гнучкості розвитку та масштабування. Незважаючи на це, існують виклики, пов'язані з потребою більш складної координації та інтеграції частин системи. Перш за все швидкість розгортання мікрофронтендів як було сказано вище зменшилась до декількох хвилин порівнюючи з монолітними системами розгортання яких може займати години це є великим аргументом в контексті використання даної архітектури, а менеджмент технологій та прийняття рішенн прискорились на 43% якщо звертати увагу на зменшення story points в оцінках програмістів завдяки ізольованному середовищу та принципу єдиної відповідальності за конткретний мікрофронтенд. У світлі порівняння мікрофронтендів з монолітними архітектурами, цей аналіз розкриває глибину і багатогранність відмінностей між цими двома підходами. З одного боку, мікрофронтенди відкривають нові горизонти в гнучкості розвитку та масштабуванні, надаючи розробникам свободу у виборі технологій та способів реалізації. Ця гнучкість виявляється в здатності швидко адаптуватися до змінних вимог ринку та користувацьких потреб, дозволяючи ефективно впроваджувати нові функціональні можливості без необхідності переробки всієї системи.

З іншого боку, монолітні структури, хоч і володіють певними перевагами у термінах спрощеності управління та деплою, часто зазнають затруднень із швидкістю внесення змін та відгуком на нововведення. Вони можуть бути надійними та ефективними у вузько визначених рамках, але їхній централізований характер може обмежувати швидкість і гнучкість розвитку.Так, мікрофронтенди виявляються значно більш ефективними у вирішенні завдань, які вимагають швидкого масштабування та адаптації до змін. Це особливо помітно у проектах, де інновації та часті зміни є ключовими. Розподілена природа

мікрофронтендів забезпечує можливість впровадження нових компонентів і функцій без значного перероблення чи ризику для стабільності всієї системи.

Проте, важливо визнати і виклики, що супроводжують мікрофронтендну архітектуру. Більш складна координація та інтеграція різних частин системи вимагає від розробників глибокого розуміння інтерфейсів взаємодії та управління залежностями. Це, у свою чергу, ставить підвищені вимоги до планування проекту та технічного дизайну, вимагаючи від команди більшої уваги до деталей інтеграції.

У підсумку, порівняльний аналіз підкреслює, що мікрофронтенди, хоч і не є універсальним рішенням для всіх типів проектів, пропонують значні переваги у тих сценаріях, де швидкість розвитку, гнучкість та можливість масштабування є ключовими.

4.1.2 Оцінка ефективності інтеграції мікрофронтендів

У цьому розділі акцентується увага на практичних прикладах інтеграції мікрофронтендів та узагальненому аналізі впливу цієї інтеграції на ефективність розробки веб-застосунків.

На практиці, інтеграція мікрофронтендів демонструє різноманітні сценарії використання. Наприклад, у великих проектах, де команди працюють над окремими модулями, мікрофронтенди дозволяють розробникам оновлювати та тестувати окремі частини застосунку незалежно одна від одної. Це знижує ризики, пов'язані з розгортанням, і сприяє швидшому внесенню змін.

Інший приклад - використання мікрофронтендів для створення персоналізованих користувацьких інтерфейсів. Завдяки модульній структурі мікрофронтендів, можна легко адаптувати інтерфейс для різних груп користувачів, інтегруючи специфічні для них компоненти. На основі аналізу практичних прикладів можна зробити висновок, що інтеграція мікрофронтендів позитивно впливає на ефективність розробки веб-застосунків. Це особливо

важливо для проектів, які потребують високої масштабованості та гнучкості. Мікрофронтенди сприяють більшій незалежності розробників, що в свою чергу призводить до швидшого циклу розробки, знижує залежності між командами та сприяє інноваціям.

Тим не менш, існують виклики, пов'язані з інтеграцією мікрофронтендів, зокрема, управління залежностями та координація між різними модулями. Важливим аспектом є також забезпечення консистентності користувацького інтерфейсу та збереження високої продуктивності в умовах інтеграції різноманітних компонентів. В цілому, інтеграція мікрофронтендів виявляється ефективним підходом до розробки сучасних веб-застосунків, забезпечуючи необхідну гнучкість та масштабованість для відповіді на зростаючі вимоги ринку та користувачів.

Оцінка ефективності інтеграції мікрофронтендів та аналіз використання Module Federation як ключового інструменту для цієї інтеграції є важливою частиною розуміння загальної картини впровадження мікрофронтендної архітектури. Module Federation, як частина архітектури мікрофронтендів відіграє ключову роль у забезпеченні гнучкості та ефективності інтеграції. Ця технологія дозволяє різним командам розробників працювати над окремими частинами застосунку незалежно, інтегруючи їх безпосередньо в рантаймі. Це забезпечує значну гнучкість та знижує залежності між різними частинами системи, що сприяє швидшому циклу розробки та легшому впровадженню нововведень.

Існують інші підходи та інструменти для інтеграції мікрофронтендів, такі як iFrames, Webpack 5, SystemJS, та інші. Однак, Module Federation вирізняється завдяки своїй здатності динамічно завантажувати мікрофронтенди та обмінюватися функціональністю між ними без необхідності для заздалегідь визначеного контракту. Це дозволяє створювати більш масштабовані та гнучкі застосунки, де зміни в одній частині системи не призводять до необхідності переробки або перекомпіляції інших частин. Вибір на користь Module Federation

у Personio був зроблений через кілька ключових факторів: Динамічне Завантаження: Module Federation дозволяє динамічно та ефективно завантажувати необхідні частини застосунку, що підвищує загальну продуктивність та швидкість відгуку системи. Незалежність Розробки: Цей підхід дозволяє різним командам працювати над своїми частинами застосунку незалежно, сприяючи паралельності робіт та швидкості реалізації проектів.Гнучкість та Масштабованість: Використання Module Federation забезпечує високу масштабованість та адаптивність застосунків, дозволяючи легко інтегрувати нові компоненти та функціональність.

Таблиця 5

Порівняння бібліотек інтеграцій мікрофронтендів

Критерій	iFrames	Webpack 5	SystemJS	Module Federation
Ізоляція	Висока	Залежить від конфігурації	Середня	Висока
Спільне використання ресурсів	Обмежене	Високе	Середнє	Дуже високе
Інтеграція	Низька	Висока	Середня	Висока
Гнучкість	Обмежена	Висока	Висока	Дуже висока

В даній таблиці можна проаналізувати порівняння бібліотек інтеграцій і ця інформація допомагає зрозуміти, чому Module Federation є популярним вибором

для інтеграції мікрофронтендів, зокрема завдяки його гнучкості, високому рівню інтеграції та ефективному спільному використанню ресурсів.

4.1.3 Методи підвищення продуктивності та ефективності мікрофронтендів

Хоч мікрофронтенди і є продуктивним рішенням багатьох проблем, при неправильному використанні вони також можуть мати низку ускладнень які можуть впливати на продутивнітсть додатка. Розвиваючи мікрофронтендну архітектуру, особливу увагу слід приділити підвищенню продуктивності та ефективності. Ключові методики та підходи, які можуть бути застосовані в цьому контексті, охоплюють не лише технічні аспекти, але й порівняння з альтернативними рішеннями в інших архітектурних моделях. У цій частині роботи ми занурюємося в аналіз та обговорення методів, які спрямовані на підвищення продуктивності та ефективності мікрофронтендних застосунків. Враховуючи складність та багатогранність мікрофронтендної архітектури, ми виявляємо кілька ключових областей для оптимізації. Ефективне Управління Залежностями. Одним з основних аспектів підвищення продуктивності мікрофронтендів є ефективне управління залежностями. Це включає в себе оптимізацію спільного використання бібліотек між різними мікрофронтендами, щоб уникнути надмірного завантаження однакових ресурсів. Використання таких інструментів, як Module Federation в Webpack, дозволяє досягти більш ефективного спільного використання та уникнення дублювання коду.

Ліниве Завантаження та Розділення Коду. Ліниве завантаження (Lazy Loading) та розділення коду є важливими техніками для покращення часу завантаження застосунків. Це передбачає завантаження лише тих компонентів та модулів, які необхідні користувачу в певний момент часу, замість завантаження всього застосунку одразу. Такий підхід забезпечує швидший початковий запуск та кращий загальний досвід користувача. Кешування та Оптимізація Ресурсів. Ефективне кешування статичних ресурсів, таких як CSS, JavaScript та

зображення, може істотно покращити продуктивність застосунків. Використання CDN (мереж доставки контенту) та оптимізація розмірів файлів також внесуть свій вклад у підвищення швидкості завантаження та реакції застосунків.

Моніторинг та Аналіз Продуктивності. Неперервний моніторинг та аналіз продуктивності застосунків є критично важливими для виявлення та вирішення проблем. Використання інструментів, таких як DataDog або Lighthouse, для відстеження ключових показників продуктивності (KPIs) дозволяє командам швидко реагувати на будь-які виявлені проблеми та оптимізувати продуктивність.

Таким чином Методи, спрямовані на підвищення продуктивності та ефективності мікрофронтендів, охоплюють широкий спектр технік і підходів, від технічних рішень, таких як управління залежностями та оптимізація коду, до стратегічних ініціатив, таких як моніторинг та аналіз продуктивності. Застосування цих методів вимагає глибокого розуміння архітектури мікрофронтендів та здатності адаптувати їх до конкретних вимог та умов проекту.

4.2 Результати роботи мікрофронтендних застосунків у реальних умовах

Досліджуючи результати впровадження мікрофронтендної архітектури у реальних умовах, особливо в контексті досвіду компанії Personio, ми відкриваємо для себе цілий спектр унікальних викликів та успіхів.

Цей аналіз дозволяє глибше зрозуміти, як мікрофронтенди впливають на різні аспекти веб-застосунків – від продуктивності до користувацького досвіду. В Personio значна увага приділялася підвищенню продуктивності за допомогою мікрофронтендів. Наприклад, інтеграція нових функціональних модулів стала швидшою та ефективнішою. Раніше процес додавання нових функцій міг тривати кілька тижнів, але з впровадженням мікрофронтендів цей час скоротився до

декількох днів. Це стало можливим завдяки автономній природі мікрофронтендів, яка дозволяє розробникам працювати над окремими частинами застосунку незалежно один від одного. У Personio було виявлено, що впровадження мікрофронтендів позитивно вплинуло на користувацький досвід. Завдяки оптимізації завантаження та рендерингу сторінок, час відгуку системи помітно зменшився, що підвищило загальну задоволеність користувачів. Реакція на користувацькі запити стала швидшою, а навігація по застосунку – інтуїтивно зрозумілішою.

Однак, впровадження мікрофронтендів у Personio також супроводжувалося певними викликами, особливо в аспектах інтеграції та управління залежностями. Наприклад, синхронізація роботи різних команд та управління спільними ресурсами вимагали ретельного планування та координації. Однак, за допомогою правильної стратегії та інструментів, таких як Module Federation, ці виклики були успішно подолані. Загалом, впровадження мікрофронтендної архітектури в Personio продемонструвало важливість цього підходу у створенні сучасних, гнучких та ефективних веб-застосунків. Попри виклики, пов'язані з інтеграцією та управлінням, переваги, такі як підвищення продуктивності, гнучкість розвитку та покращення користувацького досвіду, роблять цей підхід вартим застосування в різноманітних проектах.

4.2.2 Оцінка стабільності та надійності системи при різних навантаженнях

У рамках компанії Personio, що обслуговує понад 10 000 користувачів по всьому світу, стабільність та надійність системи є критично важливими аспектами. Різноманітність функціональності, від оплати та бухгалтерії до оптимізації праці та відповідності законодавству Європи та Америки, ставить перед системою високі вимоги до стабільності та ефективності.

Одним із ключових елементів забезпечення стабільності в Personio є розгорнута система тестування, що включає юніт та синтетичне тестування. Юніт-тести дозволяють перевірити окремі компоненти системи, гарантуючи їх надійність та коректність роботи. Синтетичне тестування, з іншого боку, імітує реальні користувацькі сценарії, дозволяючи оцінити поведінку системи в різних умовах та під різними навантаженнями.

Ще одним важливим аспектом у підтримці стабільності є здатність системи швидко відновлюватися після інцидентів. У Personio це досягається за допомогою механізмів швидкого відкочування, які дозволяють повернутися до стабільної версії системи у випадку виявлення критичних помилок або проблем.

Неперервний моніторинг системи та оперативна відповідь на інциденти є ще одним ключовим фактором, що забезпечує високу стабільність та надійність. Використання інструментів для моніторингу дозволяє команді Personio своєчасно виявляти та усувати будь-які проблеми, забезпечуючи безперебійну роботу системи.

Таким чином враховуючи велику кількість користувачів та складність функціоналу, Personio демонструє високий рівень стабільності та надійності своїх мікрофронтендних застосунків. Це досягається завдяки ефективній системі тестування, механізмам швидкого відкочування та ретельному моніторингу, які забезпечують стійкість системи в реальних умовах експлуатації.

4.2.3 Можливості для оптимізації та покращення масштабованості

Розглядаючи реалії та виклики, пов'язані з мікрофронтендними архітектурами, особливо в контексті великих систем, таких як в Personio, ми можемо ідентифікувати ряд можливостей для оптимізації та покращення масштабованості. Ці можливості охоплюють як технічні, так і організаційні аспекти, і відіграють критичну роль у забезпеченні ефективності та здатності системи адаптуватися до змінних вимог та обсягів даних.

На архітектурному рівні, оптимізація мікрофронтендів може включати рефакторинг існуючих компонентів для забезпечення більшої модульності та незалежності. Це може сприяти кращому розділенню відповідальності та гнучкості у внесенні змін без необхідності переробки великих частин системи. Одним з ключових аспектів покращення продуктивності є оптимізація процесу завантаження та виконання мікрофронтендів. Це включає застосування таких технік, як ліниве завантаження та розділення коду, що дозволяє знизити початкове навантаження та прискорити час завантаження сторінок.

У міру зростання обсягу даних та користувачів, важливо забезпечити масштабованість бекенд-сервісів та API. Оптимізація запитів до сервера, кешування даних та ефективне управління ресурсами можуть істотно покращити загальну продуктивність та швидкість відгуку системи.

Впровадження автоматизованих інструментів для моніторингу та аналізу системи дозволяє виявляти потенційні проблеми та вузькі місця на ранніх етапах. Це включає використання інструментів для моніторингу продуктивності, аналізу використання ресурсів та відстеження відгуку системи.

Таким чином, оптимізація та покращення масштабованості мікрофронтендів великих систем, таких як в Personio, вимагає комплексного підходу, що включає архітектурні зміни, технічні оптимізації та покращення процесів управління. Зосередження на цих аспектах дозволяє не тільки підвищити продуктивність та ефективність системи, але й забезпечує її готовність до масштабування у відповідь на зростаючі вимоги та обсяги даних.

4.3 Висновки з Розділу 4

1. Аналіз впровадження мікрофронтендної архітектури у веб-застосунках підтвердив значне покращення у гнучкості та ефективності розробки. Незалежна робота команд відкриває нові перспективи для швидкого впровадження інновацій та адаптації до змінних вимог ринку.

2. Результати роботи мікрофронтендів у реальних умовах демонструють їх здатність до забезпечення високої швидкодії та масштабованості. Було виявлено, що стабільність та надійність системи при різних навантаженнях значно підвищуються.

3. Можливості для оптимізації та покращення масштабованості були розглянуті та аналізовані, що дозволяє вибудовувати ефективні стратегії для майбутнього розвитку мікрофронтендних застосунків.

4. У загальному підсумку цього розділу наголошується на невід'ємній ролі інновацій та необхідності адаптації до передових технологій у створенні високопродуктивних веб-застосунків. Водночас, стає зрозуміло, що успіх у цій області не обмежується лише технічними аспектами. Він також залежить від здатності задовольняти правові стандарти та вимоги, а також від уваги до деталей, які формують досвід користувачів. Зосередження на вибудовуванні балансу між інноваційними рішеннями, дотриманням нормативних рамок і забезпеченням високого рівня задоволеності користувачів виринає як фундаментальний принцип для досягнення ефективності в сучасному світі веб-розробки.

ВИСНОВКИ

1. Динамічність контенту в мікрофронтендах відіграє ключову роль у відповідності сучасним вимогам до веб-додатків. Мікрофронтенди дозволяють розробникам створювати більш адаптивні та відповідні до потреб користувача інтерфейси. Такий підхід забезпечує можливість швидкої зміни контенту без перезавантаження всього додатку, що особливо важливо для додатків із високою взаємодією з користувачем у реальному часі.

2. Взаємодія мікрофронтендів з сервером здійснюється через мікросервіси, що сприяє ефективному розділенню функціоналу та оптимізації запитів. Це забезпечує високу гнучкість та масштабованість системи, дозволяючи підтримувати високу продуктивність застосунку. Така структура дозволяє легко вносити зміни та оновлення в окремі частини системи без впливу на інші компоненти.

3. Взаємодія між мікрофронтендами вимагає чітких механізмів комунікації для ефективної синхронізації стану між різними компонентами системи. Це включає в себе різноманітні техніки, такі як обмін подіями, спільне використання стану та API-взаємодії. Такий підхід дозволяє підтримувати консистентність даних та забезпечувати плавну взаємодію між різними частинами застосунку.

4. Автономність команд у мікрофронтендній архітектурі сприяє інноваційності та швидкості розвитку продукту. Команди можуть працювати над своїми модулями незалежно, що дозволяє швидше реагувати на зміни вимог та впроваджувати нові функції. Такий підхід

сприяє більшій гнучкості у процесі розробки та кращій адаптації до змінюваних потреб ринку.

5. Інтеграція та розгортання мікрофронтендів мають ряд викликів, особливо в інтеграції різних технологічних стеків та координації між командами. Процеси CI/CD відіграють важливу роль у забезпеченні стабільності та якості коду. Важливо мати ефективні процеси та інструменти для гарантування, що зміни в одному мікрофронтенді не призведуть до проблем в інших частинах системи.

6. Практичний досвід компанії Personio з мікрофронтендами демонструє переваги такого підходу у гнучкості та швидкості розвитку. Незалежна робота команд над окремими компонентами мікрофронтендів в Personio сприяла ефективній інноваційності та дозволила швидко адаптуватися до змінних потреб бізнесу. Це також полегшило управління складними проектами, зменшуючи залежності та ускладнення, які часто супроводжують монолітні системи.

7. Порівняння мікрофронтендів з монолітними архітектурами показує, що мікрофронтенди пропонують кращу масштабованість та гнучкість. Вони дозволяють окремим командам розробників працювати над різними функціональними частинами застосунку, вносячи зміни незалежно один від одного. Це на відміну від монолітних архітектур, де зміни у одній частині застосунку можуть вимагати великих зусиль по перевірці та адаптації всієї системи. Однак, мікрофронтенди можуть вносити додаткову складність у управління та координацію, особливо в контексті забезпечення консистентності досвіду користувача та інтеграції з бекендом.

8. Технологічна гнучкість мікрофронтендів відкриває можливості для використання різних технологічних стеків у межах одного застосунку. Наприклад, в Personio окремі команди можуть обирати

технології, які найкраще відповідають їхнім конкретним потребам та задачам, що збільшує швидкість розвитку та інноваційність. У монолітних системах весь застосунок зазвичай базується на одному технологічному стеку, що може обмежувати можливості вибору та адаптації до нових технологій.

9. Оптимізація продуктивності та навантаження в мікрофронтендах є важливою, оскільки кожен мікрофронтенд може бути оптимізований для конкретних завдань і потреб користувача. В Personio це дозволило зменшити час завантаження та покращити загальну продуктивність застосунку. У монолітних системах оптимізація продуктивності може бути складнішою, оскільки будь-які зміни мають враховувати весь застосунок в цілому.

10. Виклики інтеграції та розгортання в мікрофронтендах включають забезпечення сумісності різних компонентів та координації між командами. В Personio це вимагало ретельного планування та реалізації ефективних процесів CI/CD, щоб забезпечити, що зміни в одному компоненті не призведуть до непередбачених проблем в інших частинах системи.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Fowler M. Microfrontends. 2020. URL: https://martinfowler.com/articles/micro-frontends.html (дата звернення: 05.09.2023).
2. Geers D. Micro Frontends in Action. Manning Publications. 2020. 320 с.
3. Jackson M., Head A. Learning React: A Hands-On Guide to Building Web Applications Using React and Redux. 2nd ed. Pearson Education. 2020. С. 45-67, 142-160.
4. Wieruch R. The Road to React: Your journey to master plain yet pragmatic React.js. 2020. С. 101-120.
5. Seshadri A. Learning TypeScript 2.x: Develop and maintain captivating web applications with ease. 2nd ed. Packt Publishing. 2018. С. 84-105.
6. Стоян С., Беккерс Я. Front-End Architecture for Design Systems: A Modern Blueprint for Scalable and Sustainable Websites. O'Reilly Media. 2019. С. 60-80.
7. Kent C. Dodds. Epic React: React Fundamentals. 2021. URL: https://epicreact.dev/ (дата звернення: 15.09.2023).
8. Subramanian B. Learning React Query: Build and Refactor a React Application with React Query. Packt Publishing. 2021. С. 35-55.
9. Tanay P. Fullstack React: The Complete Guide to ReactJS and Friends. 2nd ed. Fullstack.io. 2020. С. 110-135.
10. Mead A. The Complete React Developer Course (w/ Hooks and Redux). 2020. URL: https://www.udemy.com/course/react-2nd-edition/ (дата звернення: 20.09.2023).
11. Basarat A. TypeScript Deep Dive. 2020. URL: https://basarat.gitbook.io/typescript/ (дата звернення: 25.09.2023).

12. Williams J. Introduction to Next.js. 2020. URL: https://nextjs.org/learn/basics/create-nextjs-app (дата звернення: 03.10.2023).
13. Yavorsky S. Understanding Next.js: The React Framework. 2021. С. 22-40.
14. DataDog Team. Monitoring Modern Frontend: How to Track and Analyze Your Frontend Application Performance. DataDog. 2021. URL: https://www.datadoghq.com/blog/frontend-monitoring/ (дата звернення: 10.10.2023).
15. Hamedani M. The Ultimate Guide to JavaScript Frameworks. Code with Mosh. 2020. URL: https://codewithmosh.com/p/ultimate-guide-to-javascript-frameworks (дата звернення: 17.10.2023).
16. Rauch G., Lichter S. Next.js Quick Start Guide: Server-side rendering done right. Packt Publishing. 2019. С. 12-28.
17. Cantrill A. React Query: It’s Time to Break up with Your "Global State" – Use React Query. 2020. URL: https://tkdodo.eu/blog/react-query (дата звернення: 20.10.2023).
18. Натан С., Томас Д. Advanced React Patterns. Frontend Masters. 2021. URL: https://frontendmasters.com/courses/advanced-react-patterns/ (дата звернення: 25.10.2023).

Декларація
академічної доброчесності
здобувача ступеня вищої освіти ЗНУ

Я, Лазарєв Вадим Андрійович, студент 2 курсу, денної форми навчання, Інженерного навчально-наукового інституту ім. Ю.М. Потебні ЗНУ, спеціальність 121 Інженерія програмного забезпечення, адреса електронної пошти p314154265@gmail.com,

- підтверджую, що написана мною кваліфікаційна робота на тему: «**Аналіз та дослідження мікрофронтендних архітектур на основі розробки високонавантажених веб-застосунків**» відповідає вимогам академічної доброчесності та не містить порушень, що визначені у ст. 42 Закону України «Про освіту», зі змістом яких ознайомлений;
- заявляю, що надана мною для перевірки електронна версія роботи є ідентичною її друкованій версії;
- згоден на перевірку моєї роботи на відповідність критеріям академічної доброчесності у будь-який спосіб, у тому числі за допомогою Інтернет - системи, в також архівування моєї роботи у базі даних цієї системи.

Дата 30.11.2023 Підпис _____ Лазарєв Вадим Андрійович
(студент)

Дата 30.11.2023 Підпис _____ Попівщий Василь Іванович
(науковий керівник)