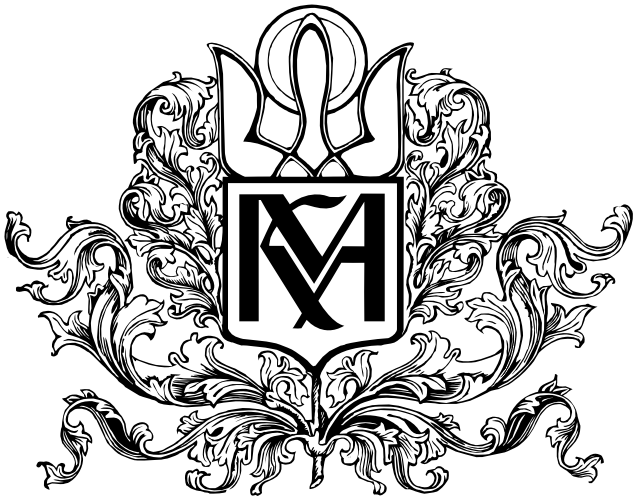
Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра інформатики факультету інформатики



ЗАСТОСУВАННЯ МІКРОФРОНТЕНДНОЇ АРХІТЕКТУРИ НА ПРИКЛАДІ ХМАРНОЇ СИСТЕМИ MATHLEARNING

Дипломний проект

на здобуття ступеня магістр

за спеціальністю «121. Інженерія програмного забезпечення»

Виконав:

студент 2-го року навчання

Тернавський Роман Павлович _____
(підпис)

Керівник:

д.ф.-м.н., проф. Малашонок Г. І. _____
(підпис)

Київ – 2024 року

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра інформатики факультету інформатики

ЗАТВЕРДЖУЮ

Зав. кафедри інформатики,

Доктор фіз.-мат. наук,

_____ Гороховський С. С.
(підпис)

«____» _____ 2024 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на дипломний проект

студенту 2 р. н. магістерської програми Інженерія програмного забезпечення Тернавському Роману Павловичу

Дослідити Застосування мікрофронтендної архітектури на прикладі хмарної системи MathLearning

Зміст текстової частини до магістерської роботи:

Перелік скорочень, умовних позначень та термінів
Анотація
Вступ
1 Огляд сучасних підходів до побудови фронтенду
2 Аналіз технологій
3 Проектування системи
Висновки
Перелік посилань

Дата видачі:

«____» _____ 2024 р.

Керівник:

д.ф.-м.н., проф. Малашонок Г. І. _____
(підпис)

Завдання отримав:

Тернавський Р. П. _____
(підпис)

Тема: Застосування мікрофронтендної архітектури на прикладі хмарної системи MathLearning

Календарний план виконання роботи:

№ п/п	Назва етапу дипломного проекту (роботи)	Термін виконання етапу	Примітка
1	Отримання завдання на дипломну роботу	28.10.2023	
2	Огляд технічної літератури за темою роботи	29.10.2023 - 30.11.2023	
3	Виконання аналізу сучасних рішень	01.12.2023 - 31.12.2023	
4	Розробка архітектури фронтенду	01.01.2024 - 15.01.2024	
5	Проектування мікрофронтендів	16.01.2024 - 31.01.2024	
6	Реалізація основних мікрофронтендів	01.02.2024 - 29.02.2024	
7	Інтеграція мікрофронтендів	01.03.2024 - 31.03.2024	
8	Тестування фронтенд-системи	01.04.2024 - 15.04.2024	
9	Оптимізація та виправлення помилок	16.04.2024 - 30.04.2024	
10	Підготовка документації	01.05.2024 - 10.05.2024	
11	Огляд та корекція документації	11.05.2024 - 16.05.2024	
12	Попередній захист	17.05.2024	
13	Остаточний захист	10.06.2024	

Керівник:

д.ф.-м.н., проф. Малашонок Г. І. _____
(підпис)

Студент:

Тернавський Р. П. _____
(підпис)

РЕФЕРАТ

Пояснювальна записка містить 81 с., 0 рис., 0 табл., 0 ліст., 0 додатків, 0 джерел

МІКРОФРОНТЕНД, ANGULAR, NX, MODULE FEDERATION, АВТОМАТИЗОВАНЕ ТЕСТУВАННЯ, ПРОДУКТИВНІСТЬ, БЕЗПЕКА, ОСВІТНЯ ПЛАТФОРМА, ХМАРНА СИСТЕМА, MATHLEARNING.

Дипломна робота на тему "Хмарна система MathLearning: фронтенд" присвячена дослідженню та реалізації мікрофронтенд архітектури для освітньої платформи з вивчення математики. У роботі використано Angular 17 та Nx монорепозиторій для впровадження мікрофронтенд архітектури з використанням Module Federation. Основні мікрофронтенди включають Shell, Account, School та Task, кожен з яких взаємодіє з відповідними бекенд сервісами. Проведено аналіз сучасних технологій, що забезпечують високу продуктивність та безпеку системи. Особлива увага приділена інтеграції мікрофронтенд модулів та використанню спільних бібліотек для підвищення ефективності розробки. Робота містить рекомендації щодо подальшого розвитку та впровадження системи у навчальні заклади.

ABSTRACT

Explanatory note contains 81 p., 0 fig., 0 table, 0 listing, 0 applications, 0 sources.

MICRO-FRONTEND, ANGULAR, NX, MODULE FEDERATION, AUTOMATED TESTING, PERFORMANCE, SECURITY, EDUCATIONAL PLATFORM, CLOUD SYSTEM, MATHLEARNING.

The master's thesis titled "Cloud System MathLearning: Frontend" is dedicated to the research and implementation of micro-frontend architecture for an educational platform focused on learning mathematics. The thesis utilizes Angular 17 and the Nx monorepository to implement micro-frontend architecture using Module Federation. The main micro-frontends include Shell, Account, School, and Task, each interacting with corresponding backend services. The analysis of modern technologies ensures high system performance and security. Special attention is given to the integration of micro-frontend modules and the use of shared libraries to enhance development efficiency. The thesis provides recommendations for further development and implementation of the system in educational institutions.

ЗМІСТ

Перелік скорочень, умовних позначень та термінів

API (Application Programming Interface) – інтерфейс прикладного програмування. Сукупність правил і механізмів для взаємодії різних програмних компонентів.

CDK (Component Development Kit) – набір інструментів для розробки компонентів. В Angular використовується для створення спільних компонентів та директив.

CI/CD (Continuous Integration/Continuous Deployment) – безперервна інтеграція/безперервне розгортання. Практика розробки програмного забезпечення, що включає автоматичне тестування та розгортання коду.

CLI (Command Line Interface) – інтерфейс командного рядка. Спосіб взаємодії з комп'ютерними програмами за допомогою текстових команд.

CSR (Client-Side Rendering) – рендеринг на стороні клієнта. Техніка, при якій HTML-сторінка рендериться на стороні клієнта за допомогою JavaScript.

HTTP (Hypertext Transfer Protocol) – протокол передачі гіпертексту. Основний протокол для передачі даних у мережі Інтернет.

JSON (JavaScript Object Notation) – формат обміну даними, заснований на синтаксисі JavaScript. Використовується для передачі даних між сервером та клієнтом.

JWT (JSON Web Token) – стандарт відкритого доступу для передачі JSON об'єктів між сторонами в закодованому вигляді.

Nx – інструментарій для монорепозиторіїв, який забезпечує управління проектами, розширеними можливостями для Angular, React та інших фреймворків.

OAuth – стандартний протокол авторизації, що дозволяє стороннім сервісам доступ до ресурсів користувача без передачі паролів.

PII (Personally Identifiable Information) – особиста ідентифікаційна інформація. Дані, що дозволяють ідентифікувати конкретну особу.

REST (Representational State Transfer) – архітектурний стиль для розробки веб-сервісів. Використовує HTTP методи для взаємодії з ресурсами.

RxJS (Reactive Extensions for JavaScript) – бібліотека для роботи з асинхронними програмами за допомогою спостережуваних потоків.

SEO (Search Engine Optimization) – оптимізація для пошукових систем. Процес покращення видимості веб-сторінки в результатах пошукових систем.

SPA (Single Page Application) – односторінковий додаток. Веб-додаток або веб-сайт, що завантажується як єдина HTML-сторінка і динамічно оновлюється при взаємодії користувача.

SSR (Server-Side Rendering) – рендеринг на стороні сервера. Техніка, при якій HTML-сторінка рендериться на сервері і надсилається клієнту в готовому вигляді.

SaaS (Software as a Service) – програмне забезпечення як послуга. Модель доставки програмного забезпечення, в якій програма розміщується на серверах постачальника і надається користувачам через Інтернет.

TDD (Test-Driven Development) – розробка через тестування. Методологія розробки програмного забезпечення, в якій тестові сценарії пишуться до написання коду.

UI (User Interface) – користувацький інтерфейс. Частина програмного забезпечення, з якою взаємодіє користувач.

UX (User Experience) – користувацький досвід. Загальний досвід і задоволення користувача від використання продукту або системи.

XML (eXtensible Markup Language) – розширювана мова розмітки. Використовується для зберігання і передачі даних у форматі, що легко читається людиною і машиною.

Анотація

Тема дослідження, представлена в даній магістерській роботі, обертається навколо застосування архітектури мікрофронтенд у розробці хмарних освітніх платформ. Основною метою є підвищення масштабованості, гнучкості та ефективності управління проектами розробки великих систем. Архітектура мікрофронтенд дозволяє розділяти фронтенд на менші, незалежні частини, кожна з яких може бути розроблена, тестована та розгорнута незалежно одна від одної. Це сприяє паралельній роботі різних команд та впровадженню інновацій з більшою швидкістю.

У рамках роботи було реалізовано декілька ключових модулів навчальної платформи, які демонструють можливості використання мікрофронтенд, включаючи модулі адміністрування, викладача, та учня. Окрема увага приділяється аспектам інтеграції цих модулів з однорідним бекендом та взаємодії через API.

Дослідження показало, що застосування архітектури мікрофронтенд не тільки сприяє модулярності та незалежності розробки, але й зменшує час затримки при внесенні змін у систему, поліпшуючи тим самим якість продукту та задоволення кінцевих користувачів. В роботі надані рекомендації щодо оптимізації процесів розробки та впровадження мікрофронтенд архітектури в масштабних проектах.

Завдяки використанню архітектури мікрофронтенд, вдалося значно підвищити гнучкість системи у відповіді на зміни та запити з боку користувачів. Кожен мікрофронтенд працює як автономний модуль із власним стеком технологій, що дозволяє командам обирати найбільш адекватні рішення для конкретних завдань без ризику негативного впливу на інші частини системи. Це не лише спрощує процес розробки та тестування, але й забезпечує вищу оперативність впровадження нововведень та оптимізацій. Крім того, ця архітектура сприяє ефективнішій балансуванні навантаження та зниженню залежності від окремих сервісів, підвищуючи загальну надійність та доступність освітньої платформи.

Вступ

Сучасні освітні системи вимагають гнучких та масштабованих рішень для забезпечення якісного навчання школярів. В умовах швидкого розвитку технологій та зростання обсягів інформації, особливої актуальності набуває створення хмарних платформ, що дозволяють учням зручно та ефективно вивчати різні предмети, зокрема математику. Одним із перспективних підходів до реалізації таких систем є використання архітектури мікрофронтенд, яка дозволяє розподілити функціональність між незалежними модулями, що полегшує їх розробку, тестування та обслуговування.

Мікрофронтенд архітектура, що застосовується у веб-розробці, дозволяє створювати складні системи з використанням окремих, незалежних компонентів. Це сприяє підвищенню масштабованості, гнучкості та зменшенню часу виходу продукту на ринок. Використання Angular та Nx монорепозиторію разом з технологією Module Federation забезпечує ефективне управління проектом, дозволяючи спростити інтеграцію та підтримку окремих частин системи.

Зростаюча потреба в персоналізованому підході до навчання також підкреслює важливість хмарних рішень, які можуть адаптуватися до індивідуальних потреб учнів. Завдяки мікрофронтенд архітектурі, освітні платформи можуть легко оновлювати та додавати нові функції без впливу на загальну стабільність системи. Це особливо важливо в умовах швидкої еволюції освітніх технологій та постійного впровадження нових методик навчання.

Застосування Angular та Nx монорепозиторію дає змогу ефективно організувати роботу над великим проектом, поділяючи його на окремі частини, які можна розробляти та тестувати незалежно. Це забезпечує більш високу якість коду, оскільки кожен мікрофронтенд модуль має чітко визначені функції та інтерфейси. Крім того, використання Module Federation дозволяє легко інтегрувати ці модулі, забезпечуючи злагоджену роботу всієї системи.

Метою даної дипломної роботи є дослідження можливостей та переваг використання мікрофронтенд архітектури для побудови клієнтської частини хмарної

системи MathLearning, призначеної для вивчення математики школярами. Для досягнення цієї мети були поставлені наступні завдання:

1. Проаналізувати сучасні підходи до побудови фронтенду та визначити місце мікрофронтенд архітектури серед них.
2. Вивчити особливості та переваги використання Angular та Nx монорепозиторію для побудови мікрофронтенд архітектури.
3. Дослідити технологію Module Federation та її застосування у контексті мікрофронтенд архітектури.
4. Розробити архітектуру системи MathLearning з використанням Angular, Nx та Module Federation.
5. Реалізувати основні компоненти системи MathLearning та забезпечити їх інтеграцію.
6. Провести тестування розробленої системи та оцінити її ефективність.

Завдання дослідження передбачають глибоке вивчення існуючих технологій та методологій, а також їх практичне застосування для створення ефективної та масштабованої освітньої платформи. У процесі роботи буде проведено аналіз сучасних підходів до розробки фронтенду, зокрема тих, що базуються на мікрофронтенд архітектурі. Особлива увага приділяється Angular та Nx, як інструментам, що дозволяють ефективно організувати розробку великих проектів.

У рамках дослідження буде також розглянуто Module Federation як засіб для забезпечення взаємодії між окремими модулями. Це включає в себе детальне вивчення механізмів інтеграції та способів забезпечення безшовної роботи різних компонентів системи. На практиці будуть реалізовані основні функціональні модулі системи MathLearning, що дозволить оцінити ефективність обраних підходів та технологій.

1 Огляд сучасних підходів до побудови фронтенду

1.1 Монолітна архітектура

Монолітна архітектура фронтенду, що колись була невід'ємною частиною веб-розробки, представляє собою підхід, де всі компоненти веб-додатку тісно інтегровані в єдину кодову базу. Це означає, що інтерфейс користувача, логіка додатку, управління станом, виклики до API та інші елементи нерозривно пов'язані між собою, створюючи монолітну структуру.

Такий підхід надає розробникам низку переваг, особливо на початкових етапах створення проекту. По-перше, це спрощує початок розробки. Оскільки всі компоненти знаходяться в одному місці, налаштування та інтеграція стають легшими та швидшими. Розробники можуть зосередитися на функціональності, не витрачаючи час на складні налаштування взаємодії між окремими модулями.

По-друге, єдине середовище розробки та розгортання зменшує складність управління проектом. Всі залежності, конфігурації та інструменти знаходяться в одному місці, що робить процес розробки більш прозорим та передбачуваним. Це особливо важливо для невеликих команд або проектів з обмеженими ресурсами.

По-третє, монолітна архітектура сприяє цілісності коду. Оскільки всі компоненти знаходяться в одному проекті, легше відстежувати залежності, виправляти помилки та забезпечувати узгодженість між різними частинами додатку. Це особливо актуально на ранніх етапах розробки, коли відбувається активне формування архітектури та функціональності.

Крім того, монолітна архітектура дозволяє легко використовувати спільні ресурси, такі як стилі, компоненти та утиліти. Це зменшує дублювання коду, підвищує його повторне використання та спрощує підтримку проекту в довгостроковій перспективі.

Однак, з розвитком проекту та збільшенням його складності, монолітна архітектура може почати виявляти свої недоліки. Масштабування окремих компонентів стає складним завданням, оскільки будь-які зміни вимагають

розгортання всього додатку. Це може призвести до простоїв, збільшення ризиків та ускладнення процесу оновлення.

Розгортання також стає повільнішим, оскільки кожна зміна вимагає повного циклу збирання та розгортання всього додатку. Це може суттєво уповільнити процес розробки та впровадження нових функцій, особливо у великих проектах.

Підтримка коду також стає складнішою. Зі збільшенням розміру кодової бази стає важче розуміти взаємозв'язки між компонентами, виявляти та виправляти помилки. Це може призвести до зниження якості коду та збільшення часу, необхідного для внесення змін.

У великих командах може виникнути невизначеність щодо розподілу відповідальності за різні частини кодової бази. Це може призвести до конфліктів, дублювання зусиль та зниження ефективності роботи команди. [1]

Лістинг 1.1 – Приклад монолітної архітектури

```
// app.module.ts
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import { HeaderComponent } from './header/header.component';
import { FooterComponent } from './footer/footer.component';
import { DashboardComponent } from './dashboard/dashboard.component';

@NgModule({
  declarations: [
    AppComponent,
    HeaderComponent,
    FooterComponent,
    DashboardComponent
  ],
  imports: [
    BrowserModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

У цьому прикладі всі компоненти інтегровані в один модуль AppModule і розгортаються разом. Всі частини додатку (заголовок, підвал, головна сторінка) знаходяться в одному кодовому базисі, що полегшує початкову розробку, але ускладнює масштабування та підтримку.

Монолітна архітектура фронтенду є потужним інструментом, але його ефективність залежить від конкретного контексту проекту. Для невеликих проектів та початкових етапів розробки вона надає низку переваг, таких як простота розробки, єдине середовище та цілісність коду. Вона дозволяє швидко розпочати роботу, спрощує координацію в невеликих командах та забезпечує високу швидкість розробки на початкових етапах.

Однак, для великих та складних проектів, особливо з великими командами, вона може стати обмежуючим фактором. Складнощі з масштабуванням, повільне розгортання та складність підтримки можуть суттєво уповільнити розвиток проекту та збільшити його вартість. У таких випадках варто розглянути альтернативні підходи, такі як мікрофронтенд архітектура, яка забезпечує більшу гнучкість та масштабованість.

Тому вибір архітектури фронтенду повинен бути ретельно обдуманим та базуватися на конкретних вимогах та обмеженнях проекту. Необхідно враховувати розмір проекту, складність функціональності, розмір команди та інші фактори, щоб зробити оптимальний вибір, який забезпечить успішну реалізацію проекту.

1.2 Серверний рендеринг (SSR)

Серверний рендеринг (SSR) є одним з найбільш ефективних підходів до побудови веб-додатків, який дозволяє значно покращити продуктивність та SEO. SSR полягає в тому, що HTML-сторінка рендериться на сервері та надсилається до клієнта в уже готовому вигляді. Це скорочує час початкового завантаження сторінки, оскільки користувач отримує повністю рендерену сторінку з самого початку. Окрім того, SSR поліпшує індексацію сторінок пошуковими системами, оскільки вони отримують готовий HTML-код замість JavaScript.

Серверний рендеринг має багато переваг. По-перше, він покращує продуктивність, оскільки HTML рендериться на сервері, і користувач отримує готову сторінку швидше. Це особливо важливо для повільних мереж або мобільних пристроїв. По-друге, SSR оптимізує сайт для пошукових систем. Пошукові системи

краще індексують сторінки з повним HTML-кодом, що підвищує видимість сайту у пошукових результатах і може призвести до збільшення трафіку. По-третє, соціальний обмін стає ефективнішим. Коли сторінка рендериться на сервері, мета-теги для соціального обміну (наприклад, Open Graph) будуть доступні відразу, що покращує вигляд попереднього перегляду при поширенні посилань у соціальних мережах. Нарешті, продуктивність на слабких пристроях також покращується. Оскільки рендеринг виконується на сервері, клієнтські пристрої можуть мати менше навантаження, що забезпечує кращу продуктивність навіть на старих або менш потужних пристроях.

Розглянемо детальніше процес реалізації серверного рендерингу на прикладі Angular додатку з використанням Angular Universal. Angular Universal забезпечує підтримку SSR для Angular додатків. Для створення нового Angular додатку з підтримкою SSR спочатку потрібно створити базовий Angular додаток, а потім додати Angular Universal.

Angular Universal створює базову конфігурацію для SSR з використанням Express.js як сервера. Після додавання Angular Universal створюється файл server.ts, який служить точкою входу для серверного рендерингу. Цей файл містить основний код для запуску Express сервера, налаштування рендерингу Angular додатку та обробки маршрутизації.

Лістинг 1.2 – Приклад налаштування SSR

```
import 'zone.js/dist/zone-node';
import { ngExpressEngine } from '@nguniversal/express-engine';
import * as express from 'express';
import { join } from 'path';
import { APP_BASE_HREF } from '@angular/common';
import { existsSync } from 'fs';
import { AppServerModule } from './src/main.server';

const app = express();
const PORT = process.env.PORT || 4000;
const DIST_FOLDER = join(process.cwd(), 'dist/angular-ssr-
example/browser');

app.engine('html', ngExpressEngine({
  bootstrap: AppServerModule,
}));

app.set('view engine', 'html');
app.set('views', DIST_FOLDER);

app.get('*.*', express.static(DIST_FOLDER, {
  maxAge: '1y'
}));

app.get('*', (req, res) => {
  res.render('index', { req, providers: [{ provide: APP_BASE_HREF,
useValue: req.baseUrl }] });
});

app.listen(PORT, () => {
  console.log(`Node server listening on http://localhost:${PORT}`);
});
```

Цей код налаштовує Express сервер для обробки запитів до Angular додатку. Спочатку імпортуються необхідні модулі, включаючи Angular Universal та Express. Потім створюється експрес додаток, налаштовується механізм рендерингу для Angular додатку та обробка статичних файлів. У кінці налаштовується обробка всіх інших запитів, які рендерять головний HTML файл додатку.

Після налаштування серверного рендерингу необхідно переконатися, що додаток правильно конфігурується для роботи з Angular Universal. Це включає налаштування серверного модуля додатку та оновлення Angular CLI конфігурації для підтримки SSR. Серверний модуль зазвичай містить лише основні компоненти та маршрути додатку, які необхідні для початкового рендерингу на сервері.

Після налаштування всіх необхідних конфігурацій можна запускати додаток з підтримкою SSR. Це дозволяє отримати всі переваги серверного рендерингу,

включаючи швидке початкове завантаження сторінки, покращену SEO оптимізацію та зручний соціальний обмін. При цьому, Angular Universal забезпечує гнучкість та можливість налаштування додатку відповідно до специфічних вимог проекту.

Серверний рендеринг (SSR) є потужним підходом до побудови сучасних веб-додатків, який дозволяє значно покращити продуктивність, SEO та загальний користувацький досвід. Використання SSR забезпечує швидке початкове завантаження сторінки, що є критично важливим для мобільних користувачів та користувачів з повільними мережами. Окрім того, SSR покращує видимість сайту у пошукових системах, що може призвести до збільшення трафіку та покращення рейтингу сайту.

Angular Universal надає потужні інструменти для реалізації SSR в Angular додатках. Він дозволяє легко налаштувати серверний рендеринг, забезпечуючи плавну інтеграцію з існуючим Angular кодом та інфраструктурою. Використання Angular Universal забезпечує всі переваги SSR, зберігаючи при цьому гнучкість та можливість налаштування додатку відповідно до потреб проекту.

Сучасні веб-додатки потребують високої продуктивності та оптимізації для пошукових систем, і серверний рендеринг є одним з найефективніших підходів для досягнення цих цілей. Використання SSR дозволяє створювати швидкі, інтерактивні та SEO-оптимізовані додатки, які забезпечують відмінний користувацький досвід.

У підсумку, серверний рендеринг є важливим інструментом для будь-якого розробника, який прагне створювати високоякісні веб-додатки. Використання Angular Universal для реалізації SSR в Angular додатках забезпечує всі необхідні інструменти та можливості для досягнення цієї мети. Це робить Angular Universal важливим компонентом сучасної веб-розробки, який дозволяє створювати продуктивні, SEO-оптимізовані та зручні у використанні додатки.

1.3 SPA (Single Page Application) архітектура

SPA (Single Page Application) — це сучасний підхід до розробки веб-додатків, який забезпечує більш плавний та інтерактивний користувацький досвід. Замість традиційного перезавантаження сторінок, SPA завантажує весь необхідний контент

один раз, а потім динамічно оновлює його за допомогою JavaScript. Цей підхід дозволяє створювати додатки, що нагадують нативні мобільні програми, з миттєвою реакцією на дії користувача та плавними переходами між різними розділами.

Основний принцип SPA полягає у використанні клієнтського рендерингу. Це означає, що більшість логіки додатку та відображення контенту відбувається на стороні користувача, використовуючи потужності його пристрою. Це знижує навантаження на сервер, оскільки не потрібно постійно відправляти запити на отримання нових сторінок. Замість цього, додаток оновлює лише необхідні частини інтерфейсу, що значно прискорює роботу та зменшує обсяг переданих даних.

Асинхронні запити (AJAX або Fetch API) є ключовим елементом SPA. Вони дозволяють додатку обмінюватися даними з сервером у фоновому режимі, не перериваючи роботу користувача. Це дає змогу динамічно оновлювати контент, підвантажувати нові дані або надсилати інформацію на сервер без необхідності перезавантаження сторінки.

SPA додатки мають односторінкову структуру, де весь контент відображається на одній сторінці, яка динамічно оновлюється. Це створює враження нативного додатку та забезпечує більш інтуїтивну взаємодію з користувачем. Навігація між різними розділами відбувається шляхом зміни вмісту сторінки, а не завантаження нових HTML-документів. [2]

Ще однією важливою особливістю SPA є локальне управління станом. Стан додатку, тобто всі дані та змінні, що визначають його поточний стан, зберігаються на стороні клієнта. Це дозволяє уникнути зайвих запитів до сервера та забезпечити більш швидку реакцію на дії користувача. Наприклад, при переході між розділами додаток може зберегти поточний стан та відновити його при поверненні, що створює більш плавний та безперервний користувацький досвід.

SPA підхід має численні переваги. Швидкість та покращений користувацький досвід (UX) є одними з найважливіших. Відсутність повного перезавантаження сторінки робить додаток більш чуйним та інтерактивним. Локальне кешування даних зменшує кількість запитів до сервера та прискорює завантаження контенту. SPA також

дозволяють створювати більш складні та динамічні інтерфейси, що робить їх привабливими для користувачів.

Лістинг 1.3 – Приклад реалізації SPA додатку

```
// app.module.ts
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { RouterModule, Routes } from '@angular/router';
import { AppComponent } from './app.component';
import { HomeComponent } from './home/home.component';
...;

const routes: Routes = [
  { path: '', component: HomeComponent },
  { path: 'about', component: AboutComponent },
  ...,
];

@NgModule({
  declarations: [AppComponent, HomeComponent, ...],
  imports: [BrowserModule, RouterModule.forRoot(routes)],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }

// app.component.ts
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  template: `<nav>
              <a routerLink="/">Home</a>
              <a routerLink="...">...</a>
            </nav>
            <router-outlet></router-outlet>`
})
export class AppComponent { }

// home.component.ts
import { Component } from '@angular/core';

@Component({
  selector: 'app-home',
  template: `<h1>Welcome to the Home page!</h1>`
})
export class HomeComponent { }
```

У цьому прикладі ми створили SPA додаток з трьома компонентами (HomeComponent, AboutComponent, ContactComponent) та маршрутизацією для навігації між ними без перезавантаження сторінки. Весь контент завантажується динамічно, що забезпечує швидкий відгук на дії користувача і покращує користувацький досвід.

SPA архітектура є потужним інструментом для створення сучасних веб-додатків, що забезпечують високу швидкість, інтерактивність та покращений користувацький досвід. Вона ідеально підходить для односторінкових додатків, інтерактивних веб-додатків та складних проектів, де важлива динамічна взаємодія з користувачами. SPA підхід дозволяє створювати додатки, що нагадують нативні мобільні програми, з плавними переходами та миттєвою реакцією на дії користувача.

Однак, вибір SPA підходу вимагає врахування його особливостей та обмежень. Необхідно приділити увагу оптимізації для пошукових систем, оскільки динамічний контент може бути складним для індексації. Також важливо оптимізувати початкове завантаження, оскільки завантаження всіх ресурсів одразу може зайняти деякий час. Крім того, збільшення кількості клієнтського коду може підвищити ризики безпеки, тому розробникам необхідно приділяти особливу увагу захисту даних та валідації.

Використання технологій серверного рендерингу (SSR) та статичної генерації сайтів (SSG) може допомогти вирішити деякі з цих проблем, забезпечуючи компроміс між швидкістю та SEO-оптимізацією. Загалом, SPA архітектура є перспективним напрямком у веб-розробці, що дозволяє створювати високоякісні та зручні додатки, які відповідають сучасним вимогам користувачів та бізнесу. Правильне використання цього підходу та врахування його особливостей допоможе розробникам створювати успішні проекти, які задовольнять потреби найвибагливіших користувачів.

1.4 Мікрофронтенд архітектура

Мікрофронтенд архітектура є сучасним підходом до розробки фронтенду, який пропонує революційний спосіб створення та управління великими веб-додатками. Замість традиційного монолітного підходу, де весь фронтенд являє собою єдиний, тісно пов'язаний код, мікрофронтенд розбиває додаток на незалежні модулі, кожен з яких відповідає за певну функціональність. [3]

Уявіть собі великий веб-магазин, де кожна сторінка - каталог товарів, кошик, оформлення замовлення - це окремий мікрофронтенд. Кожен з цих модулів може бути

розроблений, протестований та розгорнутий незалежно від інших. Це відкриває перед розробниками безліч можливостей.

По-перше, мікрофронтенди забезпечують неперевершену гнучкість. Кожен модуль може бути написаний на різних технологіях, що дозволяє використовувати найкращі інструменти для конкретних завдань. Наприклад, сторінка каталогу може бути реалізована на React, кошик - на Vue.js, а оформлення замовлення - на Angular. Така свобода вибору дозволяє командам використовувати свої улюблені технології та експериментувати з новими підходами.

По-друге, мікрофронтенди спрощують масштабування. Якщо певна частина додатку потребує більшої обчислювальної потужності або має високе навантаження, її можна масштабувати незалежно від інших модулів. Це дозволяє ефективніше використовувати ресурси та забезпечувати високу продуктивність навіть у пікові періоди.

По-третє, мікрофронтенди прискорюють розробку та розгортання. Оскільки кожен модуль є незалежним, команди можуть працювати над ними паралельно, не заважаючи одна одній. Це дозволяє швидше впроваджувати нові функції та виправляти помилки, оскільки зміни в одному модулі не впливають на роботу інших.

По-четверте, мікрофронтенди полегшують підтримку коду. Розбиття кодової бази на менші, більш керовані частини робить код більш зрозумілим та легшим для розуміння. Це спрощує пошук та виправлення помилок, а також дозволяє новим розробникам швидше включитися в роботу над проектом.

Звичайно, мікрофронтенд архітектура має і свої виклики. Одним з них є необхідність забезпечення ефективної взаємодії між модулями. Оскільки кожен модуль працює незалежно, потрібно ретельно продумати механізми обміну даними та координації дій. Іншим викликом є забезпечення узгодженості користувацького інтерфейсу. Оскільки різні модулі можуть бути розроблені різними командами, важливо створити єдиний дизайн-систему та стандарти, щоб додаток виглядав цілісно та гармонійно. [4]

Лістинг 1.4 – Приклад мікрофронтенд архітектури

```
// shell-app.module.ts
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { AppComponent } from './app.component';

@NgModule({
  declarations: [AppComponent],
  imports: [BrowserModule],
  providers: [],
  bootstrap: [AppComponent]
})
export class ShellAppModule { }

// shell-app.component.ts
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'app-root',
  template: `<h1>Shell Application</h1>
             <div id="header"></div>
             <div id="content"></div>
             <div id="footer"></div>`
})
export class AppComponent implements OnInit {
  ngOnInit() {
    // Завантаження мікрофронтенд модулів
    this.loadMicrofrontend('header', 'http://localhost:3001/header.js');
    this.loadMicrofrontend('content', 'http://localhost:3002/content.js');
    this.loadMicrofrontend('footer', 'http://localhost:3003/footer.js');
  }

  loadMicrofrontend(elementId: string, url: string) {
    const script = document.createElement('script');
    script.src = url;
    document.getElementById(elementId).appendChild(script);
  }
}
```

У цьому прикладі ми створили оболонку (ShellApp), яка завантажує три незалежні мікрофронтенд модулі (header, content, footer) з різних серверів. Кожен модуль може розроблятися та розгортатися окремо, що забезпечує високу гнучкість та масштабованість системи.

Мікрофронтенд архітектура є потужним підходом для розробки великих і складних веб-додатків, що дозволяє розподілити функціональність на незалежні модулі. Вона забезпечує високу гнучкість, масштабованість та зручність у підтримці, що робить її ідеальним вибором для проектів з великою кількістю команд та частими оновленнями.

Мікрофронтенд архітектура ідеально підходить для:

- **Великих корпоративних додатків:** Де різні команди можуть працювати над окремими модулями одночасно, використовуючи різні технології та інструменти.
- **Проектів з розподіленими командами:** Де різні частини додатку можуть розроблятися і підтримуватися незалежно одна від одної.
- **Додатків з високими вимогами до масштабованості:** Де окремі частини системи можуть розвиватися з різною швидкістю, і потрібна можливість незалежного масштабування окремих модулів.

Проте, варто враховувати складність управління та інтеграції різних модулів, а також початкові витрати на впровадження цієї архітектури. Забезпечення узгодженості користувацького інтерфейсу між різними модулями також може вимагати додаткових зусиль. Незважаючи на ці виклики, мікрофронтенд архітектура пропонує значні переваги, які можуть істотно покращити процес розробки і підтримки великих веб-додатків. [5]

1.5 Порівняння архітектур

Для розробки сучасних веб-додатків існують різні архітектурні підходи, кожен з яких має свої переваги і недоліки. У цьому підрозділі ми порівняємо три основні архітектури фронтенду: монолітну архітектуру, SPA (Single Page Application) і мікрофронтенд архітектуру. Це порівняння допоможе зрозуміти, які архітектурні рішення найкраще підходять для різних типів проектів.

Монолітна архітектура

Монолітна архітектура передбачає створення веб-додатку як єдиного цілого, де всі компоненти інтегровані в один кодовий базис. Це традиційний підхід, який був широко розповсюджений на ранніх етапах розвитку веб-технологій.

Основні характеристики:

- Єдина кодова база
- Тісна інтеграція компонентів
- Єдине середовище розробки та розгортання

Переваги:

- **Простота розробки:** Легше почати розробку, оскільки всі компоненти знаходяться в одному проекті.
- **Єдине середовище:** Спрощене управління середовищем розробки та розгортання.
- **Цілісність коду:** Легше забезпечити узгодженість між компонентами, оскільки всі вони знаходяться в одному проекті.

Недоліки:

- **Складність масштабування:** Масштабування окремих частин системи може бути складним і неефективним.
- **Повільне розгортання:** Будь-які зміни вимагають розгортання всього додатку.
- **Важкість у підтримці:** Зі збільшенням розмірів проекту складність управління кодом та забезпечення його якості зростає.

SPA (Single Page Application)

SPA передбачає створення додатку, де весь контент завантажується на одну сторінку, а взаємодія з користувачем відбувається без перезавантаження сторінок.

Основні характеристики:

- Односторінкова структура
- Клієнтський рендеринг
- Асинхронні запити

Переваги:

- **Швидкість:** Відсутність повного перезавантаження сторінки знижує затримки і забезпечує швидкий відгук.
- **Покращений UX:** Плавна і безперервна взаємодія з додатком підвищує задоволеність користувачів.
- **Локальне кешування:** Зменшує кількість запитів до сервера і покращує продуктивність.

Недоліки:

- **SEO:** Динамічна генерація контенту ускладнює оптимізацію для пошукових систем.
- **Початкове завантаження:** Може бути тривалим через необхідність завантаження всього додатку одразу.
- **Безпека:** Збільшення кількості клієнтського коду підвищує ризики безпеки.

Мікрофронтенд архітектура

Мікрофронтенд архітектура дозволяє розподілити функціональність додатку на незалежні модулі, які можуть розроблятися, розгортатися і підтримуватися окремо.

Основні характеристики:

- Незалежність модулів
- Ясні інтерфейси
- Технологічна гнучкість
- Компонентна архітектура

Переваги:

- **Масштабованість:** Легше масштабувати окремі модулі без впливу на весь додаток.
- **Простота оновлення:** Оновлення або заміна окремих частин додатку не вимагає переробки всього проекту.
- **Розподілена розробка:** Різні команди можуть працювати над окремими модулями паралельно.
- **Покращене управління кодом:** Локалізація змін в межах окремих модулів дозволяє легше тестувати та підтримувати код.

Недоліки:

- **Складність управління:** Управління більшою кількістю модулів вимагає ефективного управління взаємодією між ними.
- **Підвищені вимоги до інтеграції:** Забезпечення безшовної роботи окремих модулів може бути складним завданням.

- **Початкові витрати:** Впровадження мікрофронтенд архітектури може вимагати значних початкових зусиль та ресурсів.
- **Узгодженість користувацького інтерфейсу:** Забезпечення узгодженості користувацького інтерфейсу між різними модулями може бути викликом.

Таблиця 1.1 – Зведена таблиця порівняння архітектур

Характеристика	**Монолітна архітектура**	**Single Page Application**	**Мікрофронтенд архітектура**
Єдина кодова база	Так	Так	Ні
Незалежність модулів	Ні	Частково	Так
Масштабованість	Низька	Висока	Висока
Простота оновлення	Низька	Середня	Висока
Розподілена розробка	Ні	Так	Так
Клієнтський рендеринг	Ні	Так	Так
SEO	Висока	Низька	Висока (залежно від реалізації)
Початкові витрати	Низькі	Середні	Високі
Складність управління	Низька	Середня	Висока
Гнучкість	Низька	Висока	Висока
Вимоги до інтеграції	Низькі	Середні	Високі
Узгодженість UI	Висока	Висока	Висока (залежно від реалізації)

Для вибору оптимальної архітектури необхідно враховувати специфіку проекту, його вимоги, масштаби, команду розробників та інші фактори. Наведемо кілька ключових факторів для вибору:

- **Масштаб проекту:** Якщо проект невеликий, то монолітна архітектура може бути достатньою. Для середніх та великих проектів варто розглянути SPA або мікрофронтенд архітектуру.
- **Розподілена команда:** Якщо над проектом працює кілька команд, які відповідають за різні частини додатку, то мікрофронтенд архітектура буде оптимальним вибором.

- **Частота оновлень:** Якщо проект потребує частих оновлень та додавання нових функцій, то мікрофронтенд архітектура забезпечить простоту оновлення окремих частин додатку.
- **SEO:** Якщо важлива оптимізація для пошукових систем, то слід враховувати обмеження SPA і, можливо, поєднувати її з серверним рендерингом або використовувати мікрофронтенд архітектуру з належною реалізацією SEO.
- **Початкові витрати:** Враховуйте початкові витрати на впровадження архітектури. Монолітна архітектура потребує менше зусиль на початковому етапі, тоді як мікрофронтенд архітектура потребує значних початкових зусиль та ресурсів.

Порівнюючи різні архітектурні підходи, можна зробити висновок, що кожна архітектура має свої переваги та недоліки, і їх вибір залежить від конкретних вимог проекту. Монолітна архітектура підходить для невеликих проектів з однорідною командою розробників і обмеженими вимогами до масштабованості. SPA підходить для додатків, де важлива швидка і інтерактивна взаємодія з користувачем, але слід враховувати обмеження SEO. Мікрофронтенд архітектура є оптимальним вибором для великих і складних проектів з розподіленими командами, що потребують високої гнучкості, масштабованості і частих оновлень.

Обрання мікрофронтенд архітектури для проекту MathLearning обумовлено необхідністю забезпечити незалежність розробки, гнучкість у використанні різних технологій, а також простоту масштабування і оновлення окремих частин додатку. Цей підхід дозволить ефективно організувати роботу над проектом, забезпечити високу якість програмного забезпечення та зручність його підтримки.

2 Аналіз технологій

2.1 Angular: особливості та нововведення

Angular, розроблений та підтримуваний Google, є потужним фреймворком для створення сучасних веб-додатків. Його гнучкість та широкий спектр функціональності роблять його ідеальним вибором для розробки як односторінкових додатків (SPA), так і складних мікрофронтенд архітектур. Angular вирізняється з-поміж інших фреймворків своєю цілісністю, він надає все необхідне для побудови повноцінного додатку, включаючи інструменти для маршрутизації, управління станом, взаємодії з сервером та багато іншого.

Однією з ключових особливостей Angular є використання TypeScript, строго типізованої надбудови JavaScript. Це забезпечує вищий рівень контролю над кодом, раннє виявлення помилок та покращену підтримку інструментів розробки. TypeScript також дозволяє використовувати передові можливості мови, такі як інтерфейси, класи та декоратори, що робить код більш структурованим та читабельним.

Angular також відзначається своєю модульністю. Додатки будуються з компонентів, які є основними будівельними блоками. Кожен компонент має свою власну логіку, шаблон та стилі, що робить їх легко переносимими та повторно використовуваними. Крім того, Angular підтримує модулі, які дозволяють групувати компоненти та інші ресурси, що полегшує організацію та управління великими проектами. [6]

Декларативні шаблони Angular спрощують створення користувацького інтерфейсу. Розробники можуть використовувати HTML для визначення структури сторінки, а потім додавати спеціальні директиви та зв'язки даних для динамічного оновлення вмісту. Це робить код більш читабельним та легшим для розуміння.

Dependency Injection (DI) є ще однією потужною особливістю Angular. DI дозволяє компонентам отримувати необхідні залежності (сервіси, дані тощо) ззовні, що робить їх більш гнучкими та легшими для тестування. Angular також надає вбудовані інструменти для тестування, що дозволяють розробникам легко створювати та підтримувати юніт-тести та інтеграційні тести.

Angular підтримує реактивне програмування через бібліотеку RxJS. Це дозволяє ефективно працювати з асинхронними операціями та потоками даних, що особливо важливо для сучасних веб-додатків, які часто взаємодіють з сервером та іншими зовнішніми джерелами даних.

Angular CLI (інтерфейс командного рядка) є незамінним інструментом для розробників. Він автоматизує багато рутинних завдань, таких як створення проектів, генерацію компонентів, збірку та розгортання додатків. Це значно прискорює та спрощує процес розробки.

В останніх версіях Angular було впроваджено низку нововведень, які ще більше розширюють його можливості. Standalone компоненти дозволяють створювати компоненти без необхідності оголошувати їх у модулях, що спрощує структуру додатків. Typed Forms API покращує роботу з формами, надаючи статичну типізацію та спрощуючи валідацію. Покращення в Component Dev Kit (CDK) додають нові компоненти та покращують існуючі, що дозволяє створювати більш багатий та інтерактивний користувацький інтерфейс. [7]

Angular також отримав нові можливості маршрутизації, такі як Deferred Loading, що дозволяє відкладати завантаження модулів до моменту їхнього фактичного використання. Це може значно покращити час завантаження додатків. Directive Composition API дозволяє комбінувати директиви для створення більш складної поведінки компонентів.

Значні покращення були внесені в Server-Side Rendering (SSR), що прискорює рендеринг та покращує інтеграцію з Angular Universal. Це особливо важливо для SEO-оптимізації та покращення продуктивності на мобільних пристроях. Angular CLI також отримав оновлення, включаючи підтримку останніх версій TypeScript та інших інструментів.

Серед нових функцій варто виділити:

- **Standalone компоненти, директиви та pipes:** це дозволяє створювати їх без необхідності оголошувати їх у модулях, що спрощує структуру додатків та зменшує кількість шаблонного коду.

- **Typed Forms API:** покращує роботу з формами, надаючи статичну типізацію та спрощуючи валідацію.
- **Покращення в Component Dev Kit (CDK):** додають нові компоненти та покращують існуючі, що дозволяє створювати більш багатий та інтерактивний користувацький інтерфейс.
- **Нові можливості маршрутизації:** Deferred Loading дозволяє відкладати завантаження модулів до моменту їхнього фактичного використання.
- **Directive Composition API:** дозволяє комбінувати директиви для створення більш складної поведінки компонентів.
- **Значні покращення в Server-Side Rendering (SSR):** прискорюють рендеринг та покращують інтеграцію з Angular Universal.
- **Оновлений Angular CLI:** включає підтримку останніх версій TypeScript та інших інструментів.
- **Зміна детектування за допомогою Signals:** цей новий реактивний примітив спрощує роботу з асинхронними операціями, оновленням стану та управлінням побічними ефектами в Angular-додатках. Signals дозволяють вибірково оновлювати компоненти, що може значно покращити продуктивність, особливо у великих додатках.
- **NgOptimizedImage:** ця директива оптимізує завантаження зображень, автоматично вибираючи правильний формат та розмір зображення для різних пристроїв.
- **Input Bindings for Standalone Components:** тепер можна передавати дані в standalone компоненти так само, як і в звичайні компоненти, що робить їх ще більш зручними у використанні.
- **DoBootstrap:** цей lifecycle hook спрощує запуск додатків, що використовують bootstrapping.
- **NgTemplateOutlet:** ця директива дозволяє динамічно рендерити шаблони, що робить компоненти більш гнучкими.

- **Angular DevTools:** новий набір інструментів для налагодження Angular додатків, доступний як розширення для браузера.
- **CSP (Content Security Policy):** підтримка CSP для покращення безпеки додатків.
- **Experimental ESBuild support:** Angular CLI тепер експериментально підтримує ESBuild для швидшої збірки додатків.

Angular надає цілий ряд переваг, що роблять його ідеальним інструментом для розробки мікрофронтенд додатків. Вбудована підтримка Module Federation спрощує розробку та інтеграцію мікрофронтендів, дозволяючи легко обмінюватися модулями між різними частинами додатку, що суттєво спрощує процес розробки та розгортання окремих функціональних частин.

Зонована архітектура Angular надає засоби для створення ізольованих та незалежних мікрофронтендів, що дозволяє різним командам працювати над окремими частинами додатку, не впливаючи на інші. Це сприяє паралельній розробці та підвищує гнучкість проекту.

Signals, новий реактивний примітив, спрощує роботу з асинхронними операціями та оновленням стану, що є важливим для мікрофронтендів. Це дозволяє створювати більш ефективні та чуйні додатки, які швидко реагують на зміни даних.

Angular оптимізований для швидкого рендерингу та низького часу завантаження, що критично важливо для мікрофронтенд додатків, де кожен модуль повинен завантажуватися швидко та незалежно. Модульність Angular спонукає до створення модульних додатків, що полегшує розбиття на мікрофронтенди та забезпечує кращу структуру проекту.

Використання TypeScript забезпечує високу якість коду та легкість у його підтримці, що особливо важливо у великих проектах з багатьма розробниками. Строга типізація допомагає виявляти помилки на ранніх етапах розробки та забезпечує кращу сумісність між різними модулями.

Angular включає вбудовані інструменти для тестування, що полегшує процес написання і виконання тестів для окремих компонентів і модулів. Це дозволяє забезпечити високу якість коду та впевненість у правильності роботи кожного мікрофронтенду.

Нарешті, Angular має велику спільноту розробників і добре розроблену документацію, що забезпечує доступ до великої кількості ресурсів і підтримки. Це допомагає розробникам швидко знаходити відповіді на свої запитання та вирішувати проблеми, що виникають під час розробки.

Лістинг 2.1 – Приклад налаштування Angular проекту

```
# Встановлення Angular CLI
npm install -g @angular/cli

# Створення нового Angular проекту
ng new my-microfrontend-app

# Запуск проекту
ng serve
```

Angular є потужним та універсальним фреймворком, який пропонує широкий спектр інструментів та можливостей для розробки сучасних веб-додатків. Його сильна типізація, модульність, декларативні шаблони, впровадження залежностей та вбудовані інструменти для тестування роблять його привабливим вибором для багатьох проектів.

Останні оновлення Angular, такі як standalone компоненти, Typed Forms API, покращений CDK, нові можливості маршрутизації, Signals та підтримка Module Federation, ще більше розширюють його функціональність та роблять його ідеальним інструментом для створення мікрофронтенд архітектур.

Angular не тільки спрощує розробку складних додатків, але й забезпечує високу продуктивність, масштабованість та зручність у підтримці. Завдяки великій спільноті та активній розробці, Angular постійно розвивається та вдосконалюється, що робить його одним з провідних фреймворків у світі веб-розробки.

Незалежно від того, чи ви створюєте простий односторінковий додаток або складну мікрофронтенд архітектуру, Angular надає вам всі необхідні інструменти та можливості для досягнення успіху. Його гнучкість, потужність та зручність у використанні роблять його ідеальним вибором для будь-якого проекту веб-розробки. Нові функції, такі як Signals та Module Federation, відкривають ще більше можливостей для створення інноваційних та ефективних веб-додатків.

2.2 Nx: огляд та переваги

Nx, розроблений компанією Nrwl, є інноваційним інструментом, який революціонізує підхід до управління монорепозиторіями. У світі, де програмні проекти стають все більш складними та масштабними, Nx пропонує ефективне рішення для організації коду, підвищення продуктивності та спрощення процесів розробки.

В основі Nx лежить концепція монорепозиторію, де весь код проекту, включаючи різні додатки та бібліотеки, зберігається в одному репозиторії. Це дозволяє уникнути дублювання коду, полегшує управління залежностями та сприяє кращій координації між різними командами розробників. Nx надає потужні інструменти для роботи з монорепозиторіями, включаючи автоматизовану генерацію коду, розумну перебудову проекту та оптимізоване виконання завдань.

Однією з ключових переваг Nx є його тісна інтеграція з Angular CLI. Це дозволяє розробникам використовувати знайомі команди та інструменти Angular для створення та управління Angular-додатками в рамках монорепозиторію. Nx також підтримує інші популярні фреймворки та бібліотеки, такі як React, Next.js та NestJS, що робить його універсальним інструментом для різних проектів.

Nx сприяє модульності та повторному використанню коду. За допомогою Nx можна легко створювати бібліотеки, які містять спільний код, що використовується в різних додатках. Це дозволяє уникнути дублювання коду та спрощує внесення змін, оскільки оновлення бібліотеки автоматично застосовуються до всіх додатків, які її використовують.

Nx також надає потужні інструменти для автоматизації процесів CI/CD (безперервної інтеграції та доставки). За допомогою Nx можна легко налаштувати автоматичне тестування, збірку та розгортання додатків, що значно прискорює та спрощує процес доставки програмного забезпечення.

Гнучкість Nx забезпечується завдяки підтримці розширень та плагінів. Розробники можуть легко додавати нові функції та інструменти до Nx, налаштовуючи його під конкретні потреби проекту. Це робить Nx адаптивним та масштабованим рішенням, яке може рости разом з вашим проектом.

Nx є потужним та універсальним інструментом, який змінює підхід до розробки та управління сучасними веб-додатками. Його підтримка монорепозиторіїв, інтеграція з Angular CLI, модульність, автоматизація процесів CI/CD та гнучкість роблять його незамінним інструментом для великих та складних проектів. [8]

Nx дозволяє розробникам ефективніше організовувати код, уникнути дублювання, підвищити продуктивність та спростити процеси розробки та розгортання. Завдяки своїй масштабованості та адаптивності, Nx може бути використаний для проектів будь-якого розміру та складності, від невеликих стартапів до великих корпоративних додатків.

Використання Nx не тільки покращує технічну сторону розробки, але й сприяє кращій співпраці між командами, оскільки всі працюють над одним репозиторієм, що полегшує обмін кодом та знаннями. Це, у свою чергу, підвищує ефективність роботи та прискорює процес доставки програмного забезпечення.

Nx є інструментом майбутнього, який вже сьогодні допомагає багатьом компаніям по всьому світу створювати більш якісні, масштабовані та інноваційні веб-додатки. Якщо ви шукаєте спосіб покращити свою розробку та вивести її на новий рівень, Nx може стати вашим ключем до успіху.

2.3 Module Federation: концепція та застосування

Module Federation — це інноваційна технологія, що з'явилася у Webpack 5, яка змінює правила гри у світі веб-розробки. Вона дозволяє динамічно завантажувати та

об'єднувати модулі JavaScript з різних збірок під час виконання, що відкриває нові можливості для створення складних та масштабованих веб-додатків. [9]

Основна ідея Module Federation полягає у розподілі великого додатку на менші, незалежні модулі, які можуть бути розроблені, протестовані та розгорнуті окремо. Це дозволяє різним командам працювати над різними частинами додатку паралельно, не заважаючи одна одній, що значно прискорює процес розробки та підвищує гнучкість проекту. [10]

Module Federation працює за допомогою спеціальних файлів конфігурації Webpack, які визначають, які модулі будуть доступні для інших додатків. Ці файли включають:

- **RemoteEntry.js:** основний файл входу для віддаленого модуля, який містить всі необхідні залежності та експортує модулі для хост-додатку.
- **ContainerPlugin:** плагін Webpack, який визначає, які модулі будуть доступні для інших додатків.
- **ContainerReferencePlugin:** плагін Webpack, який дозволяє завантажувати модулі з інших збірок під час виконання.

Коли додаток запускається, він завантажує необхідні модулі з інших додатків за допомогою цих файлів конфігурації. Це дозволяє створювати додатки, які складаються з модулів, розроблених різними командами, використовуючи різні технології та фреймворки.

Module Federation є ключовим елементом у реалізації мікрофронтенд архітектури. Мікрофронтенди - це підхід до розробки фронтенду, який дозволяє розділити великий додаток на менші, незалежні частини, які можуть бути розроблені та розгорнуті окремо. Кожен мікрофронтенд може бути написаний на своїй власній технології та мати свою власну команду розробників.

Module Federation дозволяє інтегрувати ці мікрофронтенди в єдиний додаток під час виконання. Це означає, що різні частини додатку можуть бути розроблені та розгорнуті незалежно, але при цьому працювати разом як єдине ціле. [11]

Основні файли конфігурації для Module Federation:

module-federation.manifest.json: Цей файл містить інформацію про віддалені модулі та їхні URL-адреси для динамічного завантаження. Він використовується для визначення місцезнаходження віддалених модулів під час виконання додатку

Лістинг 2.2 – module-federation.manifest.json

```
{
  "remotes": {
    "login": "http://localhost:4201/remoteEntry.js",
    "dashboard": "http://localhost:4202/remoteEntry.js"
  }
}
```

main.ts: Головний файл входу для Angular додатку, який ініціалізує Angular додаток та завантажує основні модулі. В цьому файлі також викликається функція для налаштування динамічного завантаження віддалених модулів.

Лістинг 2.3 – main.ts

```
import { enableProdMode } from '@angular/core';
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';
import { AppModule } from './app/app.module';
import { environment } from './environments/environment';
import { setRemoteDefinitions } from '@nrwl/angular/mf';

if (environment.production) {
  enableProdMode();
}

fetch('assets/module-federation.manifest.json')
  .then(res => res.json())
  .then(manifest => setRemoteDefinitions(manifest))
  .then(() => platformBrowserDynamic().bootstrapModule(AppModule))
  .catch(err => console.error(err));
```

module-federation.config.ts: Конфігураційний файл для налаштування Module Federation у Webpack. Він визначає віддалені модулі та експорти для хост-додатку.

remotes: Об'єкт, що визначає віддалені модулі та їх розташування.

shared: Об'єкт, що визначає спільні залежності, які можуть використовуватися різними модулями.

Лістинг 2.4 – module-federation.config.ts

```
const ModuleFederationPlugin =
require('webpack/lib/container/ModuleFederationPlugin');

module.exports = {
  output: {
    uniqueName: 'dashboard',
    publicPath: 'auto'
  },
  optimization: {
    runtimeChunk: false
  },
  resolve: {
    alias: {
      ...sharedMappings.getAliases()
    }
  },
  plugins: [
    new ModuleFederationPlugin({
      remotes: {
        login: 'login@http://localhost:4201/remoteEntry.js'
      },
      shared: {
        '@angular/core': { singleton: true, strictVersion: true,
requiredVersion: 'auto' },
        '@angular/common': { singleton: true, strictVersion: true,
requiredVersion: 'auto' },
        '@angular/router': { singleton: true, strictVersion: true,
requiredVersion: 'auto' },
        ...sharedMappings.getDescriptors()
      }
    }),
    sharedMappings.getPlugin()
  ]
};
```

webpack.config.js: Основний конфігураційний файл Webpack, який включає module-federation.config.ts та інші налаштування для збірки додатку.

Лістинг 2.5 – webpack.config.js

```
const { merge } = require('webpack-merge');
const moduleFederationConfig = require('./module-federation.config');

module.exports = merge(moduleFederationConfig, {
  // інші налаштування Webpack
});
```

Лістинг 2.6 – Приклад реалізації інтеграції Module Federation у проект

```
// Віддалений модуль (remote.module.ts)
import { NgModule } from '@angular/core';
import { RouterModule } from '@angular/router';
import { RemoteComponent } from './remote.component';

@NgModule({
  declarations: [RemoteComponent],
  imports: [RouterModule.forChild([{ path: '', component: RemoteComponent
}])],
  exports: [RemoteComponent]
})
export class RemoteModule {}

// Хост-додаток (app-routing.module.ts)
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';

const routes: Routes = [
  { path: 'login', loadChildren: () => import('login/Module').then(m =>
m.RemoteModule) },
  { path: 'dashboard', loadChildren: () =>
import('dashboard/Module').then(m => m.RemoteModule) }
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule {}
```

Module Federation - це потужний інструмент, який відкриває нові можливості для розробки веб-додатків. Він дозволяє створювати більш гнучкі, масштабовані та модульні додатки, які легше розробляти та підтримувати.

Особливо корисним Module Federation є для мікрофронтенд архітектури. Він дозволяє розробникам розбивати великі додатки на менші, більш керовані частини, що спрощує розробку та розгортання. Крім того, Module Federation дозволяє використовувати різні технології для різних частин додатку, що дає розробникам більшу свободу вибору.

Однак, Module Federation - це нова технологія, і вона все ще розвивається. Існують деякі проблеми та обмеження, які потрібно враховувати при її використанні. Наприклад, потрібно бути обережним при управлінні залежностями між модулями, щоб уникнути конфліктів. [12]

Незважаючи на ці виклики, Module Federation є перспективною технологією, яка має потенціал змінити спосіб розробки веб-додатків. Вона вже використовується

багатьма компаніями для створення складних та масштабованих додатків, і її популярність продовжує зростати.

2.4 Nx & Module Federation: особливості реалізіції

Мікрофронтенди (Micro Frontends) - це архітектурний підхід, що дозволяє розділити великі веб-додатки на незалежні, більш керовані частини. Кожен мікрофронтенд може бути розроблений, протестований та розгорнутий окремо, що значно спрощує процес розробки та підтримки складних додатків. Nx, потужний інструмент для керування монорепозиторіями, та Module Federation, технологія динамічного завантаження модулів, є ключовими інструментами для створення ефективних мікрофронтенд архітектур з Angular.

Nx спрощує розробку мікрофронтендів, надаючи інструменти для створення та управління монорепозиторіями, де різні мікрофронтенди можуть співіснувати в одному проекті. Це дозволяє розробникам легко використовувати спільні бібліотеки та компоненти, а також автоматизувати процеси збірки та розгортання. Nx також забезпечує інтеграцію з Angular CLI, що спрощує створення та налаштування мікрофронтендів на базі Angular.

Module Federation, у свою чергу, дозволяє динамічно завантажувати та інтегрувати модулі з різних мікрофронтендів під час виконання додатку. Це означає, що кожен мікрофронтенд може бути розроблений та розгорнутий незалежно, але при цьому працювати разом з іншими мікрофронтендами в єдиному додатку. Такий підхід забезпечує високу гнучкість та масштабованість, оскільки кожен мікрофронтенд може бути оновлений або замінений без необхідності перезавантаження всього додатку.

Інтеграція Nx та Module Federation дозволяє створювати мікрофронтенд архітектури з Angular, які поєднують переваги обох інструментів. Nx спрощує управління залежностями та збірку проекту, а Module Federation забезпечує динамічне завантаження та інтеграцію модулів.

За допомогою Nx можна легко створювати нові мікрофронтенди та інтегрувати їх в існуючий проект. Nx також надає інструменти для управління залежностями між

мікрофронтендами та забезпечення їхньої сумісності. Module Federation дозволяє кожному мікрофронтенду оголошувати, які модулі він надає іншим мікрофронтендам, а також визначати, які модулі він потребує від інших мікрофронтендів.

Одним з ключових аспектів інтеграції Nx та Module Federation є налаштування конфігурації Webpack. Webpack - це збирач модулів, який використовується для створення оптимізованих збірок JavaScript коду. Конфігурація Webpack для мікрофронтендів з використанням Module Federation вимагає визначення точок входу для кожного мікрофронтенду, а також налаштування завантаження віддалених модулів.

Nx спрощує цей процес, надаючи готові конфігурації Webpack для мікрофронтендів. Розробникам потрібно лише вказати, які мікрофронтенди вони хочуть використовувати, і Nx автоматично налаштує Webpack для правильної роботи з Module Federation.

Після налаштування конфігурації Webpack, мікрофронтенди можуть взаємодіяти один з одним за допомогою обміну даними та подіями. Angular надає різні механізми для цього, такі як сервіси, RxJS та інші.

Особливо цікавим є використання Module Federation для створення динамічних мікрофронтендів. Це означає, що мікрофронтенди можуть бути завантажені та інтегровані в додаток під час його виконання, а не на етапі збірки. Це відкриває нові можливості для створення більш гнучких та адаптивних додатків, які можуть змінювати свою функціональність залежно від потреб користувача або інших факторів.

Початковим кроком є створення нового Nx workspace, який буде містити всі додатки та бібліотеки. Це можна зробити за допомогою командного рядка Nx CLI. Далі створюються хост та віддалені додатки з підтримкою Module Federation.

Лістинг 2.7 – Приклад команд для ініціалізації мікрофронтенд монорепозиторію

```
# Створення нового Nx workspace, який буде містити всі додатки та бібліотеки
npx create-nx-workspace ng-mf

# Створення хост та віддалені додатки з підтримкою Module Federation
nx g @nx/angular:host apps/dashboard --prefix=ng-mf
nx g @nx/angular:remote apps/login --prefix=ng-mf --host=dashboard

# За необхідності створення спільних бібліотек та сервісів для всіх мікрофронтів
nx g @nx/angular:lib libs/shared/data-access-user
nx g @nx/angular:service user --project=data-access-user
```

Налаштування додатку "login"

1. *module-federation.config.ts:* Цей файл містить конфігурацію Module Federation для віддаленого додатку. Він визначає віддалені модулі, які будуть доступні для хост-додатку.

Лістинг 2.8 – apps/login/module-federation.config.ts

```
import { ModuleFederationConfig } from '@nx/webpack';

const config: ModuleFederationConfig = {
  name: 'login',
  exposes: {
    './Routes': 'apps/login/src/app/remote-entry/entry.routes.ts',
  },
};

export default config;
```

2. *webpack.config.ts:* Файл об'єднує конфігурацію Module Federation та інші налаштування Webpack.

Лістинг 2.9 – apps/login/webpack.config.ts

```
import { withModuleFederation } from '@nx/angular/module-federation';
import config from './module-federation.config';

export default withModuleFederation(config);
```

3. *bootstrap.ts:* Цей файл відповідає за ініціалізацію Angular додатку.

Лістинг 2.10 – apps/login/src/bootstrap.ts

```
import { bootstrapApplication } from '@angular/platform-browser';
import { appConfig } from './app/app.config';
import { RemoteEntryComponent } from './app/remote-entry/entry.component';

bootstrapApplication(RemoteEntryComponent, appConfig).catch((err) =>
  console.error(err)
);
```

4. *app.config.ts:* Файл конфігурації додатку, який містить основні налаштування.

Лістинг 2.11 – apps/login/src/app/app.config.ts

```
import { ApplicationConfig } from '@angular/core';
import { provideRouter } from '@angular/router';
import { appRoutes } from './app.routes';

export const appConfig: ApplicationConfig = {
  providers: [provideRouter(appRoutes)],
};
```

5. *app.routes.ts:* Файл, який визначає маршрути для додатку "login".

Лістинг 2.12 – apps/login/src/app/app.routes.ts

```
import { Route } from '@angular/router';

export const appRoutes: Route[] = [
  {
    path: '',
    loadChildren: () =>
      import('./remote-entry/entry.routes').then((m) => m.remoteRoutes),
  },
];
```

6. *entry.routes.ts:* Файл, який визначає маршрути для віддаленого модуля "login".

Лістинг 2.13 – apps/login/src/app/remote-entry/entry.routes.ts

```
import { Route } from '@angular/router';
import { RemoteEntryComponent } from './entry.component';

export const remoteRoutes: Route[] = [
  { path: '', component: RemoteEntryComponent },
];
```

Налаштування додатку "dashboard"

1. *module-federation.config.ts:* Цей файл містить конфігурацію Module Federation для хост-додатку.

Лістинг 2.14 – apps/dashboard/module-federation.config.ts

```
import { ModuleFederationConfig } from '@nx/webpack';

const config: ModuleFederationConfig = {
  name: 'dashboard',
  remotes: [],
};

export default config;
```

2. *module-federation.manifest.json:* Файл, який містить інформацію про віддалені модулі.

Лістинг 2.15 – apps/dashboard/src/assets/module-federation.manifest.json

```
{
  "login": "http://localhost:4201"
}
```

3. *app.routes.ts:* Файл, який визначає маршрути для додатку "dashboard".

Лістинг 2.16 – apps/dashboard/src/app/app.routes.ts

```
import { Route } from '@angular/router';
import { loadRemoteModule } from '@nx/angular/mf';
import { AppComponent } from './app.component';

export const appRoutes: Route[] = [
  {
    path: 'login',
    loadChildren: () =>
      loadRemoteModule('login', './Routes').then((m) => m.remoteRoutes),
  },
  {
    path: '',
    component: AppComponent,
  },
];
```

4. *bootstrap.ts:* Файл відповідає за ініціалізацію Angular додатку.

Лістинг 2.17 – apps/dashboard/src/bootstrap.ts

```
import { bootstrapApplication } from '@angular/platform-browser';
import { appConfig } from './app/app.config';
import { AppComponent } from './app/app.component';

bootstrapApplication(AppComponent, appConfig).catch((err) =>
  console.error(err)
);
```

5. *app.config.ts:* Файл конфігурації додатку, який містить основні налаштування.

Лістинг 2.18 – apps/dashboard/src/app/app.config.ts

```
import { ApplicationConfig } from '@angular/core';
import { provideRouter } from '@angular/router';
import { appRoutes } from './app.routes';

export const appConfig: ApplicationConfig = {
  providers: [provideRouter(appRoutes)],
};
```

Інтеграція Nx та Module Federation є потужним інструментом для створення сучасних та масштабованих веб-додатків з Angular. Вона дозволяє розробникам розділити великий додаток на менші, більш керовані частини, що спрощує розробку, тестування та розгортання. Завдяки Module Federation, мікрофронтенди можуть бути розроблені та розгорнуті незалежно, але при цьому працювати разом як єдине ціле. Nx, у свою чергу, спрощує управління залежностями та збірку проекту, а також надає інструменти для автоматизації процесів розробки.

Використання Nx та Module Federation відкриває нові можливості для створення складних та масштабованих веб-додатків з Angular. Цей підхід дозволяє розробникам використовувати різні технології та фреймворки для різних частин додатку, що забезпечує більшу гнучкість та свободу вибору. Крім того, мікрофронтенди дозволяють розподілити роботу між різними командами, що прискорює процес розробки та підвищує ефективність роботи.

Динамічні мікрофронтенди, що завантажуються під час виконання, додають ще один рівень гнучкості та масштабованості. Вони дозволяють створювати додатки, які

можуть адаптуватися до потреб користувача та змінювати свою функціональність на льоту.

Однак, використання мікрофронтендів також вимагає від розробників додаткових зусиль для забезпечення узгодженості між різними частинами додатку та управління залежностями. Проте, переваги цього підходу значно переважують його недоліки, що робить його перспективним напрямком у веб-розробці.

2.5 QA (Quality Assurance)

Quality Assurance (QA) або забезпечення якості є критично важливим аспектом розробки програмного забезпечення. Для додатку MathLearning забезпечення якості охоплює кілька ключових напрямків, включаючи тестування коду, перевірку функціональності, продуктивності, безпеки та користувацького досвіду. Впровадження належних процесів QA допомагає гарантувати, що кінцевий продукт є стабільним, надійним та відповідає вимогам користувачів.

Автоматизоване тестування є основним компонентом QA-процесів для MathLearning. Воно забезпечує швидке та ефективне виявлення помилок у коді, дозволяючи розробникам швидко їх виправляти. У MathLearning використовуються кілька типів автоматизованих тестів, включаючи юніт-тести, інтеграційні тести та end-to-end (E2E) тести. Використання Angular та Nx дозволяє значно спростити цей процес, забезпечуючи потужний інструментарій для написання та запуску тестів.

Юніт-тестування зосереджене на перевірці окремих компонентів або функцій додатку. В Angular для юніт-тестування використовується Jasmine та Karma. Jasmine є бібліотекою для написання тестів, а Karma забезпечує запуск цих тестів у різних браузерах. Застосування Nx дозволяє легко організовувати юніт-тести у великому монорепозиторії, забезпечуючи модульність та повторне використання тестових конфігурацій. Наприклад, перевірка коректності функції входу користувача може бути реалізована за допомогою юніт-тестів.

Інтеграційне тестування перевіряє взаємодію між різними модулями додатку. У MathLearning інтеграційні тести допомагають переконатися, що окремі частини системи працюють разом належним чином. Angular та Nx надають можливість

створювати інтеграційні тести, що перевіряють взаємодію між модулями, компонентами та сервісами. Наприклад, тест може перевіряти інтеграцію модуля аутентифікації з модулем управління користувачами.

End-to-End (E2E) тестування фокусується на перевірці всієї системи з точки зору користувача. Для E2E тестування в MathLearning використовується Protractor, який дозволяє автоматизувати взаємодію з додатком через браузер. E2E тести перевіряють сценарії користувача від початку до кінця, включаючи реєстрацію, вхід, виконання завдань та інші основні функції. Angular та Nx забезпечують зручний інструментарій для налаштування та запуску E2E тестів, що дозволяє легко масштабувати процес тестування та інтегрувати його у загальний процес розробки.

Лістинг 2.19 – Приклад E2E тесту з використанням Protractor

```
import { browser, by, element } from 'protractor';

describe('MathLearning App', () => {
  it('should display the login page', () => {
    browser.get('/');
    expect(element(by.css('h1')).getText()).toEqual('Login');
  });

  it('should login successfully', () => {
    element(by.css('[name="username"]')).sendKeys('testuser');
    element(by.css('[name="password"]')).sendKeys('password');
    element(by.css('[type="submit"]')).click();
    expect(browser.getCurrentUrl()).toContain('/dashboard');
  });
});
```

Тестування продуктивності є важливою частиною QA для MathLearning, оскільки воно дозволяє перевірити, як додаток поводиться під навантаженням. Використовуються інструменти такі як JMeter або Gatling для симуляції високого навантаження на систему та вимірювання її продуктивності.

Під час тестування продуктивності перевіряються такі параметри, як час відповіді серверу, стійкість до навантаження, масштабованість та загальна продуктивність додатку. Це допомагає виявити вузькі місця у продуктивності та оптимізувати код і інфраструктуру для забезпечення стабільної роботи під високим навантаженням.

Безпека є критично важливою для будь-якого веб-додатку, особливо для таких систем, як MathLearning, які обробляють конфіденційні дані користувачів. Тестування безпеки включає виявлення вразливостей, перевірку захищеності додатку від атак та забезпечення відповідності стандартам безпеки.

Використовуються різні інструменти та методології для тестування безпеки, такі як статичний аналіз коду, динамічний аналіз, тестування на проникнення (penetration testing) та автоматизовані сканери безпеки. Наприклад, інструмент OWASP ZAP може бути використаний для автоматичного сканування додатку на наявність вразливостей. Застосування Angular та Nx дозволяє легко інтегрувати безпекові перевірки у процес розробки та забезпечувати високу якість коду з точки зору безпеки.

Забезпечення якісного користувацького досвіду є важливою частиною QA для MathLearning. Тестування UX включає перевірку зручності інтерфейсу, логіки навігації, доступності та загальної задоволеності користувачів. Для цього проводяться юзабіліті-тести, опитування користувачів та аналітика використання додатку.

Під час юзабіліті-тестів користувачі виконують типові завдання у додатку, а аналітики спостерігають за їхніми діями, виявляючи можливі проблеми або незручності. Опитування користувачів допомагають отримати зворотний зв'язок щодо зручності використання та функціональності додатку. Аналітика використання додатку дозволяє відстежувати, як користувачі взаємодіють з додатком, які функції використовуються найчастіше та де виникають проблеми.

Забезпечення якості (QA) є невід'ємною частиною розробки додатку MathLearning, яка охоплює різні аспекти, включаючи автоматизоване тестування, тестування продуктивності, безпеки та користувацького досвіду. Використання Angular та Nx забезпечує потужний інструментарій для написання, організації та запуску тестів, що дозволяє значно підвищити ефективність процесу QA. Впровадження належних процесів QA дозволяє гарантувати, що кінцевий продукт

відповідає всім вимогам користувачів та стандартам якості, забезпечуючи відмінний користувацький досвід та задоволеність клієнтів.

2.6 CI/CD & Nx: безперервна інтеграція та безперервне розгортання

Nx значно підвищує ефективність CI/CD завдяки вбудованій підтримці кешування. Як локальне, так і віддалене кешування дозволяють зберігати результати виконання завдань, таких як збірка чи тестування, та повторно використовувати їх при наступних запусках. Це значно скорочує час виконання, особливо у великих проектах, де повна перезбірка може займати значну кількість часу. Nx інтелектуально визначає, які частини коду були змінені, і перебудовує лише ті частини, що залежать від цих змін.

Nx легко інтегрується з популярними CI/CD платформами, такими як GitHub Actions, Jenkins, GitLab CI, CircleCI та інші. Це дозволяє командам використовувати вже наявні інструменти та налаштування, не витрачаючи час на складні інтеграції. Nx надає готові конфігурації та приклади для різних платформ, що спрощує процес налаштування CI/CD.

Nx також дозволяє масштабувати CI/CD процеси для великих проектів з багатьма додатками та бібліотеками. Завдяки своїй архітектурі, Nx підтримує паралельне виконання завдань, що дозволяє максимально ефективно використовувати ресурси обчислювальної системи. Це особливо корисно для великих команд, де багато розробників працюють над проектом одночасно.

Крім того, Nx спрощує налаштування CI/CD завдяки інтуїтивно зрозумілим інструментам та детальній документації. Nx Console, графічний інтерфейс користувача, дозволяє розробникам легко створювати нові проекти, запускати тести, виконувати збірки та інші завдання без необхідності глибокого знання конфігураційних файлів.

Налаштування CI/CD з Nx зазвичай включає кілька кроків:

1. **Ініціалізація Nx Workspace:** створюється новий проект Nx або ініціалізується Nx в існуючому проекті. Це налаштовує базову структуру проекту та необхідні конфігурації.

2. **Налаштування інтеграції з CI/CD платформою:** вибирається CI/CD платформа (наприклад, GitHub Actions) та налаштовуються необхідні скрипти для автоматизації збірок, тестування та розгортання. Nx надає готові шаблони та приклади для різних платформ, що спрощує цей процес.
3. **Використання кешування:** включається локальне та віддалене кешування для прискорення виконання завдань. Nx автоматично кешує результати завдань, що дозволяє уникнути повторного виконання тих самих завдань при наступних запусках.
4. **Моніторинг та оптимізація:** постійний моніторинг процесів CI/CD та оптимізація налаштувань для забезпечення максимальної ефективності. Nx надає інструменти для аналізу часу виконання завдань та виявлення потенційних проблем.

Nx є незамінним інструментом для налаштування та управління CI/CD процесами у фронтенд проектах. Завдяки своїм можливостям кешування, інтеграції з популярними платформами, масштабованості та зручності у налаштуванні, Nx допомагає командам розробників автоматизувати процеси, зменшити час виконання завдань та підтримувати високу якість коду. Це робить його невід'ємною частиною сучасного процесу розробки фронтенд-додатків, забезпечуючи швидку та надійну доставку програмного забезпечення.

Nx не лише спрощує та прискорює CI/CD, але й сприяє кращій організації проекту, повторному використанню коду та підвищенню якості розробки. Завдяки Nx, команди можуть зосередитися на створенні нових функцій та покращенні користувацького досвіду, а не на рутинних завданнях збірки та розгортання.

3 Проектування системи

3.1 Опис проекту хмарної системи MathLearning

У сучасному світі освіта відіграє ключову роль у сталому розвитку суспільства, забезпечуючи людей необхідними знаннями та навичками для подолання викликів сьогодення. Технологічний прогрес, особливо в галузі розробки програмного забезпечення, відкриває нові можливості для інтеграції інновацій в освітній процес. EdTech проекти, що поєднують освіту та технології, стають все більш популярними, трансформуючи традиційні методи навчання та створюючи сучасне, ефективне навчальне середовище.

MathLearning – це яскравий приклад такого проекту, передової освітньої платформи, розробленої з метою підвищення якості шкільної освіти. Платформа використовує сучасні технології для створення інтерактивного та динамічного навчального середовища, надаючи вчителям та учням широкий спектр інструментів для ефективної взаємодії та досягнення навчальних цілей.

Мікросервісна архітектура MathLearning забезпечує гнучкість та масштабованість системи. Кожен мікросервіс відповідає за певну функцію, що дозволяє розробникам незалежно працювати над різними компонентами платформи, такими як управління студентами, формування навчальних груп, створення навчальних планів, проведення дистанційних занять та моніторинг прогресу. Такий підхід не лише спрощує розробку та підтримку системи, але й підвищує її надійність та продуктивність.

Для вчителів MathLearning пропонує комплексний набір інструментів, що охоплюють усі аспекти навчального процесу. Вчителі можуть легко керувати інформацією про студентів, формувати навчальні групи, створювати індивідуальні навчальні плани, проводити дистанційні заняття та контрольні роботи, а також відстежувати успішність кожного учня. Це дозволяє вчителям ефективно організовувати навчальний процес, адаптувати його до потреб кожного учня та забезпечувати зворотний зв'язок.

Учні, у свою чергу, отримують доступ до інтерактивних навчальних матеріалів, включаючи записані лекції, презентації та інтерактивні вправи. Вони можуть виконувати практичні завдання, проходити тести та екзамени, а також відстежувати свій прогрес у навчанні. Такий підхід сприяє активному залученню учнів у навчальний процес, підвищує їхню мотивацію та допомагає їм краще засвоювати матеріал.

MathLearning пропонує низку переваг, які роблять його цінним інструментом для сучасної освіти. Інтерактивні методи навчання, персоналізований підхід, доступність навчання з будь-якого місця та в будь-який час, а також акцент на розвитку ключових навичок, таких як критичне мислення та комунікація, роблять MathLearning потужним каталізатором для покращення якості освіти.

MathLearning – це інноваційна освітня платформа, яка має потенціал революціонізувати шкільну освіту. Завдяки використанню сучасних технологій, мікросервісної архітектури та інтерактивних методів навчання, MathLearning створює сприятливе середовище для ефективного навчання та розвитку учнів.

Платформа надає вчителям потужні інструменти для управління навчальним процесом, створення індивідуальних навчальних планів та відстеження прогресу учнів. Учні, у свою чергу, отримують доступ до інтерактивних навчальних матеріалів, практичних завдань та тестів, що допомагає їм краще засвоювати матеріал та розвивати ключові навички.

MathLearning може стати ключовим елементом у трансформації шкільної освіти, зробивши її більш доступною, інтерактивною та персоналізованою. Це інструмент, який допоможе школам підготувати учнів до викликів майбутнього, надаючи їм необхідні знання та навички для успішної кар'єри та активної участі у суспільстві.

З огляду на постійний розвиток технологій та зростаючу потребу в інноваційних освітніх рішеннях, MathLearning має всі шанси стати одним з провідних гравців на ринку EdTech. Його потенціал для покращення якості освіти та

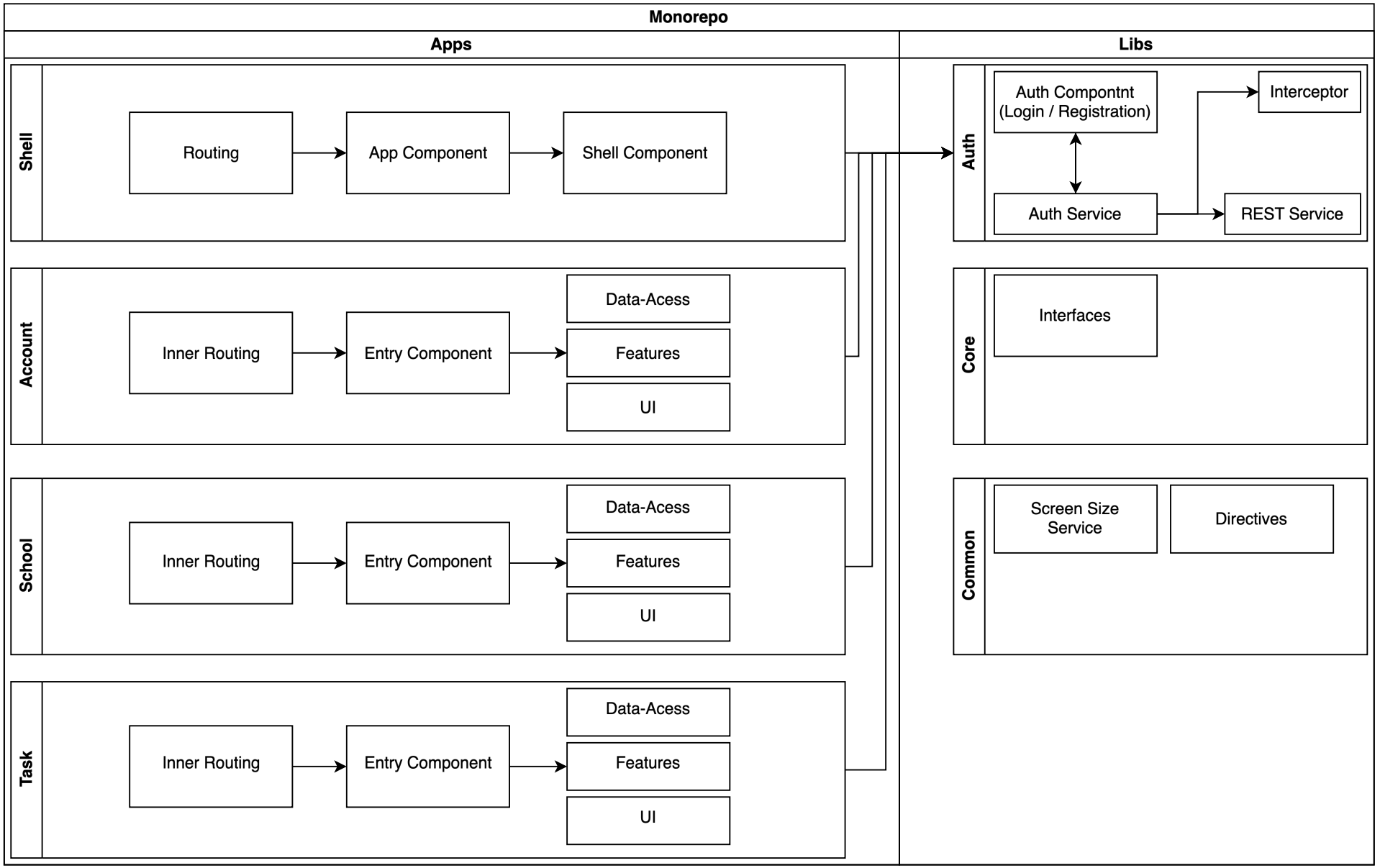
розширення можливостей учнів та вчителів є величезним, і його майбутнє виглядає дуже перспективним. [13]

3.2 Архітектура веб-інтерфейсу хмарної системи MathLearning

Архітектура системи MathLearning розроблена з урахуванням сучасних принципів модульного підходу та мікрофронтенд архітектури, що дозволяє забезпечити гнучкість, масштабованість та легкість у підтримці додатку. Основною метою проектування архітектури було створення структури, яка дозволяє незалежну розробку та розгортання окремих модулів, забезпечуючи при цьому їх взаємодію та інтеграцію в єдину систему. Для реалізації цієї мети було обрано Angular та Nx монорепозиторій, що надає ефективні інструменти для організації та управління проектом.

Давайте детально розглянемо архітектура MathLearning, включаючи її основні компоненти: хост та дистанційні мікрофронтенд модулі та бібліотеки. Кожен мікрофронтенд модуль відповідає за реалізацію окремої функціональності, такої як вивчення, тестування чи управління користувачами, що дозволяє розробляти їх незалежно один від одного.

Рисунок 3.1 – Архітектура веб-інтерфейсу хмарної системи MathLearning



3.2.1 Shell мікрофронтенд

Shell мікрофронтенд є центральним компонентом хмарної системи MathLearning, який забезпечує інтеграцію всіх інших мікрофронтендів, створюючи цілісний користувацький досвід. Його основні функції включають маршрутизацію, загальний дизайн, авторизацію та аутентифікацію користувачів, а також управління спільними компонентами, такими як заголовки сторінок і меню навігації. Shell мікрофронтенд також відповідає за забезпечення безпечного доступу до системи, інтегруючи всі інші мікрофронтенди через Module Federation.

Перш за все, Shell мікрофронтенд реалізує складну систему маршрутизації, яка дозволяє користувачам безперешкодно переміщуватися між різними частинами системи. Це забезпечується використанням Angular Router, який дозволяє динамічно завантажувати компоненти та модулі на основі маршруту. Наприклад, коли користувач переходить до розділу облікових записів, Shell мікрофронтенд завантажує відповідний Account мікрофронтенд і відповідні компоненти.

Авторизація та аутентифікація є критично важливими аспектами Shell мікрофронтенду. Використовуючи такі технології, як JSON Web Tokens (JWT), Shell забезпечує, що тільки авторизовані користувачі мають доступ до певних розділів системи. Користувачі повинні пройти через процес аутентифікації, який включає введення імені користувача та пароля, після чого система видає JWT токен, який використовується для перевірки подальших запитів. Цей процес гарантує, що всі дії користувачів відслідковуються і захищені.

Shell мікрофронтенд також забезпечує єдиний дизайн і стилі для всіх компонентів системи. Це досягається шляхом використання Angular Material або інших бібліотек UI, які забезпечують консистентний і сучасний вигляд інтерфейсу. Спільні компоненти, такі як заголовки сторінок, панелі навігації, меню та футери, створюються в Shell і використовуються іншими мікрофронтендами. Це забезпечує єдність дизайну та полегшує обслуговування системи, оскільки всі зміни в дизайні можуть бути внесені в одному місці.

Окрім цього, Shell мікрофронтенд виконує роль координатора для інших мікрофронтендів, забезпечуючи їх інтеграцію та співпрацю. Це реалізується через Module Federation, яка дозволяє динамічно завантажувати та інтегрувати мікрофронтенди в систему. Наприклад, коли користувач переходить до розділу завдань, Shell мікрофронтенд завантажує Task мікрофронтенд, який взаємодіє з Task service на бекенді для отримання та обробки даних.

Безпека є ще одним важливим аспектом Shell мікрофронтенду. Він відповідає за управління сесіями користувачів, захист від CSRF (Cross-Site Request Forgery) атак та забезпечення безпечного зберігання токенів. Використовуючи такі методи, як HTTPS та OAuth2, Shell гарантує, що всі дані користувачів передаються безпечно і конфіденційно.

Shell мікрофронтенд також забезпечує механізми для обробки помилок та моніторингу системи. Він включає глобальні обробники помилок, які дозволяють відстежувати та реєструвати помилки, що виникають у системі. Це забезпечує швидке виявлення та виправлення проблем, що покращує загальну надійність системи.

Завдяки своїй архітектурі, Shell мікрофронтенд є легко розширюваним та підтримуваним. Використання модульного підходу та поділ на мікрофронтенди дозволяє легко додавати нові функції або змінювати існуючі без значних змін в коді. Це забезпечує гнучкість і швидкість розробки, що є важливими перевагами для великих систем, таких як MathLearning.

В підсумку, Shell мікрофронтенд є центральним елементом хмарної системи MathLearning, який забезпечує інтеграцію, безпеку та єдність дизайну для всіх компонентів системи. Його роль у маршрутизації, авторизації, управлінні спільними компонентами та забезпеченні безпеки робить його критично важливим для успішної роботи всієї системи.

3.2.2 Account мікрофронтенд

Account мікрофронтенд відповідає за всі аспекти управління обліковими записами користувачів у системі MathLearning. Цей мікрофронтенд забезпечує

реєстрацію нових користувачів, автентифікацію, редагування профілів, відновлення паролів та інші функції, пов'язані з обліковими записами. Він взаємодіє з Account service на бекенді, який обробляє всі запити, пов'язані з обліковими записами.

Першим кроком для будь-якого нового користувача є процес реєстрації. Account мікрофронтенд надає зручний інтерфейс для введення необхідних даних, таких як ім'я, електронна пошта та пароль. Після заповнення форми, дані надсилаються до Account service, де вони перевіряються і зберігаються у базі даних. У випадку успішної реєстрації користувач отримує підтвердження та може увійти в систему.

Вхід до системи здійснюється через процес автентифікації, який також підтримується Account мікрофронтендом. Користувачі вводять свої облікові дані, і ці дані перевіряються на бекенді. У випадку успішної перевірки користувач отримує JWT токен, який використовується для подальшої взаємодії з системою. Цей токен зберігається в локальному сховищі браузера і автоматично додається до всіх запитів до серверу, забезпечуючи безпечний доступ до захищених ресурсів.

Account мікрофронтенд також дозволяє користувачам редагувати свої профілі. Користувачі можуть змінювати свої персональні дані, такі як ім'я, електронна пошта та пароль. Ці зміни надсилаються до Account service, де вони оновлюються в базі даних. Процес редагування профілю включає перевірку введених даних для забезпечення їх правильності та безпеки.

Відновлення паролів є важливою функцією Account мікрофронтенду. Користувачі можуть запросити відновлення паролю, ввівши свою електронну пошту. Після цього система надсилає їм лист з інструкціями та посиланням для скидання паролю. Користувачі переходять за цим посиланням, вводять новий пароль, і цей пароль зберігається у базі даних після перевірки.

Account мікрофронтенд також підтримує видалення облікових записів. Користувачі можуть вирішити видалити свій обліковий запис, і цей запит обробляється через Account service. Видалення облікового запису включає видалення особистих даних користувача з бази даних, щоб забезпечити конфіденційність і відповідність вимогам захисту даних. Важливо зазначити, що видалення облікового

запису фактично означає видалення лише приватної інформації (PII), тоді як сам обліковий запис зберігається порожнім для уникнення випадкового створення нового облікового запису з тими ж залежностями.

Завдяки використанню Angular і Nx монорепозиторію, Account мікрофронтенд є легко масштабованим і підтримуваним. Angular забезпечує потужний інструментарій для створення динамічних і інтерактивних інтерфейсів, а Nx монорепозиторій дозволяє організовувати код у модульному вигляді, що полегшує його розширення та обслуговування.

Інтерфейс Account мікрофронтенду є зручним та інтуїтивно зрозумілим. Він включає різні підказки та валідації, які допомагають користувачам правильно вводити свої дані. Використання адаптивного дизайну забезпечує коректне відображення інтерфейсу на різних пристроях, включаючи комп'ютери, планшети та мобільні телефони.

Загалом, Account мікрофронтенд є ключовим компонентом системи MathLearning, який забезпечує всі необхідні функції для управління обліковими записами користувачів. Його можливості включають реєстрацію, автентифікацію, редагування профілів, відновлення паролів та видалення облікових записів, що робить його невід'ємною частиною системи, яка забезпечує безпеку і зручність використання для всіх користувачів.

3.2.3 School мікрофронтенд

School мікрофронтенд відповідає за управління всіма аспектами шкільної структури та адміністрації в системі MathLearning. Він забезпечує функціонал для створення та управління шкільними профілями, класами, групами, навчальними планами та моніторингом прогресу студентів. Цей мікрофронтенд взаємодіє з School service на бекенді, який обробляє всі запити, пов'язані з управлінням школами та користувачами.

Однією з основних функцій School мікрофронтенду є створення та управління шкільними профілями. Користувачі, які бажають створити нову школу, можуть зробити це через інтерфейс School мікрофронтенду, заповнивши відповідну форму з

необхідною інформацією. Після створення школи, директор автоматично отримує відповідний профіль, що дозволяє йому керувати школою, створювати профілі для інших користувачів та призначати їм ролі.

Управління користувачами є ще однією важливою функцією School мікрофронтенду. Користувачі з роллю директора або заступника можуть створювати профілі для інших користувачів, таких як вчителі та студенти. Кожен профіль містить інформацію про роль користувача та іншу специфічну інформацію, яка притаманна школі. Вчителі та студенти не мають можливості створювати шкільні профілі, однак вони можуть переглядати свою інформацію та асоційовані з ними класи та групи.

Крім того, School мікрофронтенд дозволяє керувати класами та групами студентів. Класи можуть створюватися та редагуватися користувачами з роллю заступника, тоді як групи створюються та заповнюються вчителями. Ця функція дозволяє організовувати студентів у відповідні навчальні групи, що полегшує управління навчальним процесом та спостереження за прогресом.

Прив'язка навчальних планів до груп є ще однією важливою функцією School мікрофронтенду. Вчителі можуть призначати навчальні плани групам студентів, що дозволяє їм ефективно організовувати навчальний процес. Кожен навчальний план містить перелік завдань та дедлайни, що дозволяє студентам отримувати завдання вчасно та виконувати їх у визначені терміни.

Моніторинг прогресу студентів є критично важливою функцією School мікрофронтенду. Вчителі можуть переглядати прогрес кожного студента за навчальними планами та групами, до яких вони мають доступ. Ця інформація дозволяє вчителям аналізувати успішність студентів, виявляти слабкі місця та надавати додаткову підтримку при необхідності.

Використання Angular та Nx монорепозиторію забезпечує гнучкість та масштабованість School мікрофронтенду. Angular надає потужний інструментарій для створення динамічних та інтерактивних інтерфейсів, що забезпечує зручність використання для всіх користувачів. Nx монорепозиторій дозволяє організовувати код у модульному вигляді, що полегшує його розширення та обслуговування.

Інтерфейс School мікрофронтенду є зручним та інтуїтивно зрозумілим. Він включає різні підказки та валідації, які допомагають користувачам правильно вводити свої дані. Використання адаптивного дизайну забезпечує коректне відображення інтерфейсу на різних пристроях, включаючи комп'ютери, планшети та мобільні телефони.

Загалом, School мікрофронтенд є критично важливим компонентом системи MathLearning, який забезпечує всі необхідні функції для управління шкільною структурою та адміністрацією. Його можливості включають створення та управління шкільними профілями, класами, групами, навчальними планами та моніторинг прогресу студентів, що робить його невід'ємною частиною системи, яка забезпечує ефективність та зручність використання для всіх користувачів.

3.2.4 Task мікрофронтенд

Task мікрофронтенд відповідає за всі аспекти управління навчальними завданнями у системі MathLearning. Цей мікрофронтенд забезпечує створення, редагування та видалення завдань, їх розподіл по навчальних планах, а також контроль дедлайнів. Він взаємодіє з Task service на бекенді, який обробляє всі запити, пов'язані з управлінням завданнями та навчальними планами.

Однією з основних функцій Task мікрофронтенду є створення та редагування навчальних завдань. Користувачі з відповідними правами можуть створювати нові завдання, заповнюючи форму з необхідною інформацією, такою як назва, опис, тип завдання та дедлайн. Завдання можуть бути двох типів: "потребують відповіді" і "не потребують відповіді". Перший тип включає задачі, які мають умову і питання для вирішення, тоді як другий тип представляє собою лекційний матеріал.

Редагування завдань є ще однією важливою функцією Task мікрофронтенду. Користувачі можуть змінювати існуючі завдання, оновлюючи їх інформацію або додаючи новий контент. Це дозволяє підтримувати завдання актуальними та відповідними до поточних навчальних вимог. Усі зміни зберігаються у базі даних, що забезпечує їх доступність для всіх користувачів.

Видалення завдань також підтримується Task мікрофронтендом. Користувачі можуть видаляти завдання, які більше не потрібні, звільняючи місце для нових завдань. Видалення завдань включає видалення всієї інформації, пов'язаної з цими завданнями, з бази даних, що забезпечує чистоту та організованість системи.

Прив'язка завдань до навчальних планів є критично важливою функцією Task мікрофронтенду. Користувачі можуть створювати навчальні плани, які включають перелік завдань та дедлайни. Ці плани допомагають організувати навчальний процес, забезпечуючи студентам чіткі інструкції щодо виконання завдань. Навчальні плани можуть бути розподілені по групах студентів, що дозволяє вчителям ефективно керувати навчальним процесом.

Контроль дедлайнів є ще однією важливою функцією Task мікрофронтенду. Користувачі можуть встановлювати дедлайни для кожного завдання, що допомагає студентам організувати свій час та виконувати завдання у встановлені терміни. Система також може надсилати нагадування студентам про наближення дедлайнів, що допомагає їм залишатися в курсі своїх завдань.

Моніторинг виконання завдань є важливою функцією Task мікрофронтенду Користувачі можуть відстежувати прогрес виконання завдань студентами, переглядати їх відповіді та надавати зворотний зв'язок. Це дозволяє вчителям оцінювати успішність студентів, виявляти проблеми та надавати додаткову підтримку при необхідності.

Використання Angular та Nx монорепозиторію забезпечує гнучкість та масштабованість Task мікрофронтенду. Angular надає потужний інструментарій для створення динамічних та інтерактивних інтерфейсів, що забезпечує зручність використання для всіх користувачів. Nx монорепозиторій дозволяє організовувати код у модульному вигляді, що полегшує його розширення та обслуговування.

Інтерфейс Task мікрофронтенду є зручним та інтуїтивно зрозумілим. Він включає різні підказки та валідації, які допомагають користувачам правильно вводити свої дані. Використання адаптивного дизайну забезпечує коректне

відображення інтерфейсу на різних пристроях, включаючи комп'ютери, планшети та мобільні телефони.

Загалом, Task мікрофронтенд є критично важливим компонентом системи MathLearning, який забезпечує всі необхідні функції для управління навчальними завданнями. Його можливості включають створення, редагування та видалення завдань, їх розподіл по навчальних планах, контроль дедлайнів та моніторинг виконання завдань, що робить його невід'ємною частиною системи, яка забезпечує ефективність та зручність використання для всіх користувачів.

3.2.5 Спільні бібліотеки

Використання спільних бібліотек у рамках Nx монорепозиторію є ключовим елементом для забезпечення масштабованості, повторного використання коду та підтримуваності великих проектів, таких як MathLearning. У цьому проекті використовуються декілька спільних бібліотек, зокрема auth, common/cdk, та core, кожна з яких відіграє важливу роль у розробці та підтримці системи.

Бібліотека auth відповідає за реалізацію механізмів аутентифікації та авторизації у системі MathLearning. Вона включає в себе всі необхідні компоненти, сервіси та утиліти для управління процесами входу, виходу, реєстрації користувачів та перевірки їхніх прав доступу. Використання спільної бібліотеки auth забезпечує єдність та безпеку процесів аутентифікації у всіх мікрофронтендах.

AuthService є основним сервісом для управління аутентифікацією користувачів. Він включає методи для входу, виходу, реєстрації та відновлення паролів. AuthGuard виконує роль захисного механізму, який перевіряє, чи є користувач автентифікованим перед доступом до певних маршрутів або компонентів. TokenInterceptor додає токени автентифікації до HTTP-запитів, забезпечуючи безпечну взаємодію з бекендом. LoginComponent та RegisterComponent реалізують інтерфейс входу та реєстрації користувачів. Бібліотека auth значно спрощує процес управління аутентифікацією та авторизацією, забезпечуючи централізовану реалізацію та підвищуючи безпеку системи.

Бібліотека common/cdk (Component Development Kit) включає загальні компоненти, директиви та утиліти, які використовуються у різних частинах системи MathLearning. Це дозволяє уникнути дублювання коду та забезпечити консистентність у реалізації спільних функціональних можливостей.

Ця бібліотека включає UI-компоненти, такі як кнопки, форми та модальні вікна, які використовуються у різних мікрофронтендах. Директиви допомагають реалізовувати загальні функції, такі як валідація форм, обробка подій та анімації. Пайпи служать для обробки та форматування даних, наприклад, формати дати, чисел та тексту. Helper-сервіси реалізують загальні функції, такі як обробка помилок, логування та управління станом. Використання бібліотеки common/cdk дозволяє розробникам повторно використовувати загальні компоненти та утиліти, що значно підвищує ефективність розробки та підтримки коду.

Бібліотека core включає основні сервіси, модулі та конфігурації, які є критичними для функціонування всієї системи MathLearning. Вона забезпечує централізовану реалізацію основних функцій та залежностей, які використовуються у різних частинах системи.

CoreModule імпортує та експортує всі необхідні сервіси та компоненти для інших частин системи. API-сервіси відповідають за взаємодію з бекендом, наприклад, HttpClientWrapper забезпечує централізовану обробку HTTP-запитів. ConfigurationService керує конфігураціями системи, такими як налаштування API, параметри середовища та інші глобальні конфігурації. LoggerService забезпечує централізоване логування подій та помилок у системі. Бібліотека core забезпечує централізовану реалізацію основних функцій та залежностей, що значно спрощує управління конфігураціями та залежностями у системі.

Використання спільних бібліотек auth, common/cdk та core у рамках Nx монорепозиторію дозволяє забезпечити масштабованість, повторне використання коду та підтримуваність проекту MathLearning. Ці бібліотеки забезпечують централізовану реалізацію критичних функцій, що підвищує ефективність розробки

та підтримки системи, а також забезпечує консистентність та безпеку у всіх мікрофронтендах.

3.3 Дизайн веб-інтерфейсу хмарної системи MathLearning

Інтеграція мікрофронтенд модулів є одним з ключових аспектів розробки сучасних веб-додатків, особливо у випадку великих та складних систем, таких як MathLearning. Використання мікрофронтенд архітектури дозволяє розподілити відповідальність за різні частини системи між незалежними командами, забезпечуючи масштабованість, гнучкість та можливість повторного використання компонентів. У цій системі основні мікрофронтенди — Shell, Account, School та Task — інтегруються за допомогою технології Module Federation, що є частиною Webpack 5.

3.3.1 Підготовка до інтеграції

Процес інтеграції мікрофронтенд модулів починається з налаштування кожного з них як окремого модуля, який може бути завантажений динамічно. Це досягається шляхом конфігурації Webpack для використання Module Federation. Кожен мікрофронтенд налаштовується як remote module, який надає певні компоненти або сервіси для використання іншими мікрофронтендами. Важливо ретельно налаштувати експортування необхідних модулів та компонентів, щоб забезпечити їх правильну інтеграцію у систему.

Лістинг 3.1 – Приклад конфігурації Webpack для Account мікрофронтенду

```
const ModuleFederationPlugin =
require('webpack/lib/container/ModuleFederationPlugin');

module.exports = {
  // інші налаштування Webpack
  plugins: [
    new ModuleFederationPlugin({
      name: 'account',
      filename: 'remoteEntry.js',
      exposes: {
        './AccountModule': './src/app/account/account.module.ts',
      },
      shared: ['@angular/core', '@angular/common', '@angular/router'],
    }),
  ],
```

У цьому прикладі мікрофронтенд називається account, а модуль AccountModule експортований для використання іншими мікрофронтендами. Схожі конфігурації застосовуються до інших мікрофронтендів. Це дозволяє кожному мікрофронтенду мати власну незалежну структуру та бути окремим проєктом у межах загальної системи.

3.3.2 Shell як основа інтеграції

Shell мікрофронтенд виконує роль основної оболонки, яка об'єднує всі інші мікрофронтенди в єдину систему. Він відповідає за маршрутизацію, загальний дизайн, авторизацію та аутентифікацію користувачів, а також управління спільними компонентами. Shell використовує Module Federation для динамічного завантаження мікрофронтенд модулів на основі маршруту або іншої логіки. Це забезпечує гнучкість системи та дозволяє легко додавати або оновлювати окремі мікрофронтенди без впливу на інші частини системи.

Лістинг 3.2 – Приклад конфігурації Webpack для Shell мікрофронтенду

```
const ModuleFederationPlugin =
require('webpack/lib/container/ModuleFederationPlugin');

module.exports = {
  // інші налаштування Webpack
  plugins: [
    new ModuleFederationPlugin({
      name: 'shell',
      remotes: {
        account: 'account@http://localhost:3001/remoteEntry.js',
        school: 'school@http://localhost:3002/remoteEntry.js',
        task: 'task@http://localhost:3003/remoteEntry.js',
      },
      shared: ['@angular/core', '@angular/common', '@angular/router'],
    }),
  ],
};
```

У цьому прикладі Shell мікрофронтенд визначає віддалені модулі account, school та task, які завантажуються з відповідних URL. Це дозволяє Shell мікрофронтенду динамічно завантажувати ці модулі та інтегрувати їх у загальну систему. Динамічне завантаження модулів забезпечує зменшення часу завантаження

та оптимізацію продуктивності, оскільки кожен модуль завантажується лише тоді, коли це необхідно.

3.3.3 Маршрутизація та завантаження модулів

Маршрутизація в Shell мікрофронтенді налаштовується таким чином, щоб динамічно завантажувати необхідні модулі на основі маршруту. Angular Router дозволяє легко реалізувати цю функціональність за допомогою loadRemoteModule з Angular Module Federation. Це забезпечує плавну навігацію між різними мікрофронтендами, створюючи єдиний користувацький досвід. Користувачі можуть безперешкодно переходити між різними розділами системи, не помічаючи, що вони використовують різні мікрофронтенди.

Лістинг 3.3 – Приклад маршрутизації для Shell мікрофронтенду

```
import { loadRemoteModule } from '@angular-architects/module-federation';

const routes: Routes = [
  {
    path: 'account',
    loadChildren: () =>
      loadRemoteModule({
        remoteEntry: 'http://localhost:3001/remoteEntry.js',
        remoteName: 'account',
        exposedModule: './AccountModule',
      }).then(m => m.AccountModule),
  },
  {
    path: 'school',
    loadChildren: () =>
      loadRemoteModule({
        remoteEntry: 'http://localhost:3002/remoteEntry.js',
        remoteName: 'school',
        exposedModule: './SchoolModule',
      }).then(m => m.SchoolModule),
  },
  {
    path: 'task',
    loadChildren: () =>
      loadRemoteModule({
        remoteEntry: 'http://localhost:3003/remoteEntry.js',
        remoteName: 'task',
        exposedModule: './TaskModule',
      }).then(m => m.TaskModule),
  },
  // інші маршрути
];
```

У цьому прикладі, коли користувач переходить до маршруту /account, Shell мікрофронтенд завантажує AccountModule з віддаленого модуля account. Аналогічно, для маршрутів /school та /task завантажуються відповідні модулі. Це дозволяє створювати гнучку систему, яка може адаптуватися до змін вимог або розширюватися без значних змін у коді.

3.3.4 Спільні сервіси та компоненти

Для забезпечення консистентності та повторного використання коду між мікрофронтендами використовуються спільні сервіси та компоненти, які розташовані у спільних бібліотеках Nx. Це дозволяє уникнути дублювання коду та забезпечити єдину реалізацію загальних функцій. Спільні бібліотеки, такі як auth, common/cdk та core, включають в себе всі необхідні компоненти, сервіси та утиліти, які можуть бути використані різними мікрофронтендами. Наприклад, бібліотека auth містить сервіси та компоненти для управління аутентифікацією, які можуть бути використані як в Shell мікрофронтенді, так і в інших мікрофронтендах.

Бібліотека auth відповідає за реалізацію механізмів аутентифікації та авторизації у системі MathLearning. Вона включає в себе всі необхідні компоненти, сервіси та утиліти для управління процесами входу, виходу, реєстрації користувачів та перевірки їхніх прав доступу. Використання спільної бібліотеки auth забезпечує єдність та безпеку процесів аутентифікації у всіх мікрофронтендах. Наприклад, AuthService є основним сервісом для управління аутентифікацією користувачів. Він включає методи для входу, виходу, реєстрації та відновлення паролів. AuthGuard виконує роль захисного механізму, який перевіряє, чи є користувач автентифікованим перед доступом до певних маршрутів або компонентів. TokenInterceptor додає токени автентифікації до HTTP-запитів, забезпечуючи безпечну взаємодію з бекендом.

Бібліотека common/cdk (Component Development Kit) включає загальні компоненти, директиви та утиліти, які використовуються у різних частинах системи MathLearning. Це дозволяє уникнути дублювання коду та забезпечити консистентність у реалізації спільних функціональних можливостей. Ця бібліотека включає UI-компоненти, такі як кнопки, форми та модальні вікна, які

використовуються у різних мікрофронтендах. Директиви допомагають реалізовувати загальні функції, такі як валідація форм, обробка подій та анімації. Пайпи служать для обробки та форматування даних, наприклад, формати дати, чисел та тексту. Helper-сервіси реалізують загальні функції, такі як обробка помилок, логування та управління станом. Використання бібліотеки common/cdk дозволяє розробникам повторно використовувати загальні компоненти та утиліти, що значно підвищує ефективність розробки та підтримки коду.

Бібліотека core включає основні сервіси, модулі та конфігурації, які є критичними для функціонування всієї системи MathLearning. Вона забезпечує централізовану реалізацію основних функцій та залежностей, які використовуються у різних частинах системи. CoreModule імпортує та експортує всі необхідні сервіси та компоненти для інших частин системи. API-сервіси відповідають за взаємодію з бекендом, наприклад, HttpClientWrapper забезпечує централізовану обробку HTTP-запитів. ConfigurationService керує конфігураціями системи, такими як налаштування API, параметри середовища та інші глобальні конфігурації. LoggerService забезпечує централізоване логування подій та помилок у системі. Бібліотека core забезпечує централізовану реалізацію основних функцій та залежностей, що значно спрощує управління конфігураціями та залежностями у системі.

3.3.5 Управління станом та комунікація між мікрофронтендами

Управління станом та комунікація між мікрофронтендами є важливим аспектом інтеграції. Використання централізованого сховища стану дозволяє ефективно управляти глобальним станом додатку та забезпечити синхронізацію даних між різними мікрофронтендами. Для MathLearning використовується новий підхід, який застосовує Angular Signals та RxJS, що дозволяє зберігати дані реактивно та інтерактивно.

Angular Signals надає можливість створювати реактивні значення, які автоматично оновлюються при зміні їх залежностей. Це дозволяє значно спростити управління станом та зменшити кількість ручного коду для оновлення компонентів.

У поєднанні з RxJS, який забезпечує потужні інструменти для роботи з асинхронними даними та потоками подій, цей підхід стає ще більш ефективним.

Для початку, створюється централізоване сховище стану, де визначаються всі основні сигнали та потоки даних. Наприклад, сигнал для аутентифікаційної інформації користувача може бути визначений як реактивне значення, яке автоматично оновлюється при зміні стану користувача.

Лістинг 3.4 – Приклад створення сигналу для аутентифікаційної інформації

```
import { Injectable } from '@angular/core';
import { BehaviorSubject } from 'rxjs';
import { signal, computed } from '@angular/core';

@Injectable({
  providedIn: 'root'
})
export class AuthService {
  private userSubject = new BehaviorSubject<User | null>(null);
  user$ = this.userSubject.asObservable();

  userSignal = signal<User | null>(null);

  constructor() {}

  login(user: User) {
    this.userSubject.next(user);
    this.userSignal.set(user);
  }

  logout() {
    this.userSubject.next(null);
    this.userSignal.set(null);
  }

  get isAuthenticated() {
    return computed(() => !!this.userSignal.get());
  }
}
```

У цьому прикладі AuthService використовує і RxJS, і Angular Signals для управління станом користувача. userSubject та user$ забезпечують реактивність через RxJS, тоді як userSignal надає реактивне значення через Angular Signals. Методи login та logout оновлюють як BehaviorSubject, так і Signal, забезпечуючи консистентність даних. [14]

Компоненти можуть підписуватися на зміни стану через RxJS або Angular Signals залежно від конкретних потреб. Це дозволяє створювати високоефективні компоненти, які автоматично оновлюються при зміні стану.

Комунікація між мікрофронтендами здійснюється через централізоване сховище стану та загальні сервіси. Angular Signals та RxJS дозволяють легко організувати цю комунікацію, забезпечуючи синхронізацію даних між різними частинами системи.

Наприклад, коли користувач оновлює свій профіль в Account мікрофронтенді, інші мікрофронтенди можуть бути сповіщені про ці зміни через спільний сигнал або поток даних. Це забезпечує єдину точку правди для стану додатку та спрощує управління даними.

Лістинг 3.5 – Приклад використання сигналів для комунікації між мікрофронтендами

```
import { Component, OnInit } from '@angular/core';
import { AuthService } from './auth.service';
import { Subscription } from 'rxjs';

@Component({
  selector: 'app-profile',
  template: `
    <div *ngIf="user">
      <h1>Welcome, {{ user.name }}</h1>
    </div>
  `
})
export class ProfileComponent implements OnInit {
  user: User | null = null;
  private subscription: Subscription;

  constructor(private authService: AuthService) {}

  ngOnInit() {
    this.subscription = this.authService.user$.subscribe(user => {
      this.user = user;
    });

    this.user = this.authService.userSignal.get();
    this.authService.userSignal.subscribe(user => {
      this.user = user;
    });
  }

  ngOnDestroy() {
    this.subscription.unsubscribe();
  }
}
```

У цьому прикладі ProfileComponent підписується на зміни стану користувача як через RxJS, так і через Angular Signals. Це дозволяє компоненту автоматично оновлюватися при зміні стану користувача, забезпечуючи інтерактивність та реактивність.

Використання Angular Signals та RxJS надає кілька важливих переваг для управління станом та комунікації між мікрофронтендами:

- **Реактивність:** Сигнали та потоки даних автоматично оновлюються при зміні їх залежностей, що значно спрощує управління станом.
- **Масштабованість:** Централізоване сховище стану дозволяє легко масштабувати систему, додаючи нові мікрофронтенди або змінюючи існуючі без значних змін у коді.
- **Консистентність:** Використання єдиної точки правди для стану додатку забезпечує консистентність даних та поведінки додатку.
- **Продуктивність:** Реактивне управління станом дозволяє оптимізувати продуктивність, оскільки компоненти оновлюються лише тоді, коли це необхідно.

Управління станом та комунікація між мікрофронтендами за допомогою Angular Signals та RxJS забезпечують ефективне, реактивне та масштабоване рішення для великих веб-додатків, таких як MathLearning. Використання цих технологій дозволяє створювати інтерактивні та високопродуктивні додатки, які легко підтримувати та розширювати, забезпечуючи зручність та задоволення користувачів.

3.3.6 Взаємодія з бекенд сервісами

Кожен мікрофронтенд взаємодіє з відповідними бекенд сервісами для отримання та обробки даних. Наприклад, Account мікрофронтенд взаємодіє з Account service для управління обліковими записами користувачів, тоді як School мікрофронтенд взаємодіє з School service для управління шкільними профілями та групами. Використання спільних бібліотек для реалізації API сервісів дозволяє забезпечити консистентність та повторне використання коду для взаємодії з бекендом.

Лістинг 3.6 – Приклад сервісу для взаємодії з бекендом

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';
import { User } from './user.model';

@Injectable({
  providedIn: 'root'
})
export class UserService {
  private apiUrl = 'http://localhost:3000/api/users';

  constructor(private http: HttpClient) {}

  getUsers(): Observable<User[]> {
    return this.http.get<User[]>(this.apiUrl);
  }

  getUser(id: number): Observable<User> {
    return this.http.get<User>(`${this.apiUrl}/${id}`);
  }

  createUser(user: User): Observable<User> {
    return this.http.post<User>(this.apiUrl, user);
  }

  updateUser(user: User): Observable<User> {
    return this.http.put<User>(`${this.apiUrl}/${user.id}`, user);
  }

  deleteUser(id: number): Observable<void> {
    return this.http.delete<void>(`${this.apiUrl}/${id}`);
  }
}
```

У цьому прикладі сервіс UserService забезпечує взаємодію з бекендом для управління користувачами. Використання HttpClient дозволяє легко здійснювати HTTP-запити до бекенд сервісів, забезпечуючи отримання та обробку даних. Це дозволяє забезпечити консистентність та повторне використання коду для взаємодії з бекендом.

3.3.7 Висновки

Один з головних принципів мікрофронтенд архітектури — це забезпечення розширюваності та легкої підтримки системи. Використання модульного підходу дозволяє додавати нові мікрофронтенди або змінювати існуючі без значних змін в коді інших частин системи. Це забезпечує гнучкість у розробці та дозволяє швидко реагувати на зміни вимог або додавати нові функціональні можливості.

Важливим аспектом розширюваності є можливість оновлення окремих мікрофронтендів незалежно один від одного. Це дозволяє уникнути великих оновлень, які можуть вплинути на всю систему, та забезпечити безперервність роботи навіть під час впровадження нових функцій або виправлення помилок. Крім того, використання централізованого сховища стану та спільних бібліотек дозволяє зберігати консистентність даних та поведінки додатку навіть при зміні окремих мікрофронтендів.

Використання Nx монорепозиторію забезпечує гнучкість та масштабованість системи. Nx дозволяє організовувати код у модульному вигляді, що полегшує його розширення та обслуговування. Це дозволяє розробникам працювати над окремими частинами системи, не турбуючись про вплив на інші частини. Крім того, Nx забезпечує підтримку багатьох корисних інструментів та утиліт, які полегшують процес розробки та тестування.

Інтеграція мікрофронтенд модулів у системі MathLearning є складним, але надзвичайно важливим процесом, який забезпечує масштабованість, гнучкість та повторне використання коду. Використання технології Module Federation та спільних бібліотек Nx дозволяє ефективно організувати взаємодію між різними частинами системи, забезпечуючи їхню інтеграцію у єдину цілісну платформу. Це забезпечує швидкість розробки, спрощує підтримку та дозволяє легко додавати нові функціональні можливості у майбутньому.

Мікрофронтенд архітектура дозволяє командам розробників працювати незалежно одна від одної, зосереджуючись на конкретних функціональних областях. Це підвищує ефективність роботи та дозволяє швидше реагувати на зміни вимог. Крім того, розділення відповідальності за різні частини системи дозволяє зменшити ризики та покращити якість коду.

Завдяки використанню централізованого сховища стану та спільних бібліотек, система зберігає консистентність даних та поведінки додатку, що є критично важливим для великих та складних проектів. Інтеграція мікрофронтенд модулів

забезпечує надійну та ефективну платформу для реалізації системи MathLearning, яка здатна задовольнити потреби користувачів та забезпечити їх успішне навчання.

Висновки

Дипломна робота "Хмарна система MathLearning: фронтенд" зосередилася на дослідженні та впровадженні мікрофронтенд архітектури для освітньої платформи, призначеної для вивчення математики. Основною метою роботи було використання сучасних технологій, таких як Angular та Nx, для забезпечення високої продуктивності, безпеки та масштабованості системи. У цьому розділі підсумуємо основні переваги, які були отримані завдяки використанню мікрофронтенд архітектури, а також технологій Angular та Nx, і обговоримо внесок системи MathLearning у навчальний процес.

Система MathLearning є комплексною освітньою платформою, яка забезпечує інтерактивне навчання математики для школярів. Основні мікрофронтенди системи включають Shell, Account, School та Task, кожен з яких взаємодіє з відповідними бекенд сервісами. Мікрофронтенд архітектура дозволила розділити систему на незалежні функціональні частини, що забезпечило легке оновлення та масштабування системи. Це особливо важливо для освітніх платформ, де постійно з'являються нові вимоги та потреби.

Однією з основних переваг мікрофронтенд архітектури є її масштабованість. Використовуючи мікрофронтенди, ми змогли розділити додаток на менші, більш керовані частини, кожна з яких відповідає за окремий функціональний блок. Це дозволило різним командам працювати над різними частинами додатку одночасно, не заважаючи одна одній. Наприклад, команда, яка працює над мікрофронтендом Account, могла зосередитися на функціоналі управління користувачами, тоді як інша команда займалася мікрофронтендом Task, забезпечуючи розробку та оптимізацію функціоналу для завдань. Такий підхід значно покращив координацію роботи та зменшив час, необхідний для розробки нових функцій.

Мікрофронтенд архітектура забезпечила високу гнучкість розробки. Ми змогли легко інтегрувати нові функціональні можливості, змінювати існуючі компоненти та модулі без впливу на інші частини системи. Це стало можливим завдяки використанню незалежних мікрофронтендів, які можуть бути розгорнуті та оновлені

окремо. Наприклад, оновлення мікрофронтенду School не вимагало змін в інших частинах системи, що значно спрощувало процес розробки та підтримки. Це дозволило швидше реагувати на зміну вимог користувачів і адаптувати систему під нові потреби без тривалих затримок.

Мікрофронтенди дозволяють розподіляти навантаження на різні сервери або навіть на різні географічні регіони. Це забезпечує високу доступність та стійкість системи до навантажень. У випадку збою одного з мікрофронтендів, інші частини системи залишаються працездатними, що підвищує загальну надійність платформи MathLearning. Такий підхід гарантує, що навіть при високих навантаженнях користувачі отримують стабільний доступ до ключових функцій системи, що є критичним для освітнього процесу.

Мікрофронтенд архітектура також дозволила нам оптимізувати продуктивність системи шляхом розподілу функціональності між різними мікрофронтендами. Кожен мікрофронтенд міг бути оптимізований окремо, забезпечуючи швидке завантаження та високу продуктивність користувацького інтерфейсу. Це особливо важливо для освітньої платформи, де швидкість та зручність користування є критичними для забезпечення ефективного навчального процесу. Оптимізація продуктивності включала як зменшення часу завантаження, так і підвищення загальної швидкодії системи при обробці великих обсягів даних. [15]

Angular є одним з найсучасніших та найпотужніших фреймворків для розробки веб-додатків, що забезпечує високу продуктивність та зручність розробки. Використання Angular дозволило нам швидко створювати інтерфейси користувача, які є динамічними та реактивними. Завдяки широкому спектру вбудованих функцій та інструментів, Angular значно полегшує розробку складних додатків, забезпечуючи при цьому високу якість коду. Крім того, Angular забезпечує підтримку модульної архітектури, що дозволяє розробникам створювати незалежні компоненти, які можуть бути повторно використані в різних частинах додатку. Це сприяє зменшенню дублювання коду та підвищенню його підтримуваності.

Nx монорепозиторій, використаний у проекті, надав додаткові переваги в організації та управлінні кодом. Завдяки Nx, ми змогли ефективно організувати структуру проекту, розподіляючи код на незалежні бібліотеки та додатки. Це дозволило нам легко керувати залежностями та забезпечити повторне використання коду між різними мікрофронтендами. Крім того, Nx надає потужні інструменти для аналізу продуктивності, виявлення залежностей та оптимізації процесів розробки, що значно підвищило ефективність роботи команд розробників.

Використання Angular та Nx також сприяло поліпшенню якості коду та зменшенню кількості помилок. Angular надає потужну систему типізації, що дозволяє уникати багатьох типових помилок на етапі компіляції. Крім того, система модульного тестування Angular дозволяє легко створювати та виконувати юніт-тести, що забезпечує високу якість коду. Nx, у свою чергу, забезпечує інтеграцію з інструментами автоматизованого тестування та аналізу коду, що дозволяє виявляти проблеми ще до того, як вони потраплять у продакшн.

Ще однією важливою перевагою використання Angular та Nx є їх підтримка широкого спектру сучасних веб-технологій та стандартів. Це включає підтримку прогресивних веб-додатків (PWA), серверного рендерингу (SSR), а також інтеграцію з різними сервісами та API. Таке широке охоплення забезпечує гнучкість у виборі архітектурних підходів та дозволяє створювати сучасні, масштабовані та продуктивні веб-додатки.

Завдяки використанню Angular та Nx, ми змогли створити стабільну та надійну систему, яка відповідає всім вимогам сучасних веб-додатків. Висока продуктивність, масштабованість, гнучкість та зручність розробки стали основними перевагами, які ми отримали завдяки цьому вибору. Мікрофронтенд архітектура, реалізована за допомогою Angular та Nx, дозволила забезпечити високу якість коду, зменшити час розробки та підтримки, а також підвищити загальну ефективність роботи команди розробників.

Система MathLearning має великий потенціал для покращення навчального процесу. Завдяки інтерактивному та динамічному інтерфейсу, студенти можуть легко

взаємодіяти з навчальним матеріалом, виконувати завдання та відстежувати свій прогрес. Вчителі, у свою чергу, отримують інструменти для ефективного управління навчальним процесом, оцінювання та моніторингу досягнень студентів. MathLearning дозволяє створювати персоналізовані навчальні плани, що враховують індивідуальні потреби та рівень підготовки кожного студента. Це сприяє підвищенню мотивації та залученості учнів до навчання.

Використання хмарних технологій забезпечує доступ до MathLearning з будь-якого пристрою та в будь-який час, що є особливо важливим в умовах дистанційного навчання. Хмарна система забезпечує збереження та безпеку даних, дозволяючи користувачам зосередитися на навчальному процесі, не турбуючись про технічні аспекти. Крім того, хмарна інфраструктура забезпечує масштабованість системи, що дозволяє легко додавати нових користувачів та розширювати функціональні можливості платформи.

MathLearning також надає можливість інтеграції з іншими освітніми платформами та сервісами, що забезпечує зручність використання та розширює функціональні можливості системи. Інтеграція з популярними інструментами для відеоконференцій, наприклад, дозволяє проводити онлайн-уроки безпосередньо через платформу MathLearning, що робить навчальний процес більш зручним та ефективним.

У підсумку, використання мікрофронтенд архітектури, а також технологій Angular та Nx, значно покращило процес розробки та якість кінцевого продукту. Ми змогли створити сучасну, масштабовану та продуктивну систему, яка відповідає високим вимогам до освітніх платформ. Ці технології дозволили забезпечити швидке та ефективне впровадження нових функцій, оптимізувати продуктивність системи та забезпечити високу якість користувацького досвіду. Впровадження таких підходів та технологій у розробку додатку MathLearning стало важливим кроком у створенні сучасної освітньої платформи, яка відповідає потребам користувачів та вимогам ринку. MathLearning має потенціал зробити значний внесок у навчальний процес, забезпечуючи зручність, ефективність та інноваційність у вивченні математики. [2]

Перелік посилань

1. Ingeno J. Software Architect's Handbook. Packt Publishing, 2018. 594 pp.
2. Mikowski M., Powell J. Single Page Web Applications. Manning, 2013. 432 pp.
3. Peltonen S., Mezzalira L., та Taibi D. Motivations, benefits, and issues for adopting Micro-Frontends: A Multivocal Literature Review // Information and Software Technology. Gothenburg. 2021. Vol. 136.
4. Mitra R., Nadareishvili I. Microservices: Up and Running: A Step-by-Step Guide to Building a Microservices Architecture. Vol 1. O'Reilly Media, 2020. 316 pp.
5. Steyer M. Enterprise Angular: Micro Frontends and Moduliths with Angular. Leanpub, 2024. 203 pp.
6. Google. // Angular Documentation: [сайт]. [2024]. URL: https:// angular.dev/
7. Google. // Angular Material UI component library: [сайт]. [2024]. URL: https://material.angular.io/
8. Nrwl. // Nx: Extensible Dev Tools for Monorepos: [сайт]. [2024]. URL: https://nx.dev
9. Webpack. // Module Federation: [сайт]. [2024]. URL: https:// webpack.js.org/concepts/module-federation/
10. Steyer M. // Micro Frontends with Modern Angular - Part 1: Standalone and esbuild - ANGULARarchitects: [сайт]. [2023]. URL: https:// www.angulararchitects.io/en/blog/micro-frontends-with-modern-angular-part-1-standalone-and-esbuild/
11. Module Federation core team. // Module Federation: [сайт]. [2024]. URL: https://module-federation.io/

12. Steyer M. // Micro Frontends with Modern Angular - Part 2: Multi-Version and Multi-Framework Solutions with Angular Elements and Web Components - ANGULARarchitects: [сайт]. [2024]. URL: https://www.angulararchitects.io/en/blog/micro-frontends-with-modern-angular-part-2-multi-version-and-multi-framework-solutions-with-angular-elements-and-web-components/
13. Costa C., Alvelos H., та Teixeira L. The Use of Moodle e-learning Platform: A Study in a Portuguese University // Procedia Technology. 2012. Vol. 5.
14. Morony J. // Learn to build professional-grade Angular Applications | Angular Start: [сайт]. [2024]. URL: https://angularstart.com/
15. Yang C., Chuanchang L., та Zhiyuan S. Research and Application of Micro Frontends // IOP Conference Series: Materials Science and Engineering. Shanghai. 2019. Vol. 490.
16. Ford N., Parsons R., та Kua P. Building Evolutionary Architectures: Support Constant Change. Vol 1. O'Reilly Media, 2017. 188 pp.
17. Bootstrap team. // Bootstrap · The most popular HTML, CSS, and JS library in the world.: [сайт]. [2024]. URL: https://getbootstrap.com/