УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА У НОВОМ САДУ

Младен Бајић

МИКРО ФРОНТЕНД АРХИТЕКТУРА: ТЕОРИЈА И ПРИМЈЕНА У РАЗВОЈУ ВЕБ АПЛИКАЦИЈА

ДИПЛОМСКИ РАД
- Основне академске студије -

Novi Sad, 2026

UNIVERZITET U NOVOM SADU
FAKULTET TEHNIČKIH NAUKA U NOVOM SADU

Mladen Bajić

MIKRO FRONTEND ARHITEKTURA: TEORIJA I PRIMJENA U RAZVOJU VEB APLIKACIJA

DIPLOMSKI RAD
- Osnovne akademske studije -

Novi Sad, 2026

УНИВЕРЗИТЕТ У НОВОМ САДУ ● **ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА**
21000 НОВИ САД, Трг Доситеја Обрадовића 6

КЉУЧНА ДОКУМЕНТАЦИЈСКА ИНФОРМАЦИЈА

Редни број, **РБР**:		
Идентификациони број, **ИБР**:		
Тип документације, **ТД**:		Монографска документација
Тип записа, **ТЗ**:		Текстуални штампани материјал
Врста рада, **ВР**:		Дипломски рад – основне академске студије
Аутор, **АУ**:		Младен Бајић
ментор, **МН**:		др Срђан Сладојевић, редовни професор
Наслов рада, **НР**:		Микро Фронтенд архитектура: теорија и примјена у развоју веб апликација
Језик публикације, **ЈП**:		српски/ћирилица
Језик извода, **ЈИ**:		српски/енглески
Земља публиковања, **Плата**:		Република Србија
Уже географско подручје, **УГП**:		АП Војводина, Нови Сад
године, **ГО**:		2026
Издавач, **ИЗ**:		Ауторски репринт
Место и адреса, **МА**:		Нови Сад 21000, Трг Доситеја Обрадовића 6
Физички опис рада, **ФО**: (поглавља/страна/ цитата/табела/слика/листинга/прилога)		8/78/19/24/20/0
научна област, **АЛИ**:		ИМТ – информационе технологије
Научна дисциплина, **НД**:		Инжењерство информационих система
Предметна одредница/Кључне речи, **ПО**:		Микрофронтенд, веб апликације
УДЦ		
чувај се **Цху**:		Библиотека ФТН-а, Трг Доситеја Обрадовића, Нови Сад
Важна напомена, **ВН**:		
закључак, **ИЗ**:		Рад истражује микрофронтенд архитектуру кроз три приступа: *iframe*, *Webpack Module Federation* и *Single-SPA*, имплементирана на идентичној е-продавница апликацији (*B-World*) у *React*-у, *Vue.js*-у и *Svelte*-у. *Lighthouse* мјерења показала су да *Module Federation* остварује најбоље перформансе (334 KB), *iframe* пружа изолацију за старе системе, а *Single-SPA* највећу флексибилност. Резултат је матрица одлучивања за избор оптималног приступа.
Датум прихватања теме, **ДП**:		
Датум одбране, **ТО**:		
Чланови комисије, **КО**:	Председник:	Др Андраш Андерла, редовни професор
	Члан:	Др Марко Арсеновић, доцент
	Члан, ментор:	Др Срђан Сладојевић, редовни професор

Потпис ментора

UNIVERSITY OF NOVI SAD ● **FACULTY OF TECHNICAL SCIENCES**
21000 NOVI SAD, Trg Dositeja Obradovića 6

KEY WORDS DOCUMENTATION

Accession number, **ANO**:			
Identification number, **INO**:			
Document type, **DT**:		Monographic publication	
Type of record, **TR**:		Textual printed material	
Contents code, **CC**:		Bachelor thesis	
Author, **AU**:		Mladen Bajić	
Mentor, **MN**:		Srdjan Sladojevic, Ph.D.	
Title, **TI**:		Micro Frontend architecture: theory and application in web application development	
Language of text, **LT**:		Serbian	
Language of abstract, **LA**:		Serbian/English	
Country of publication, **CP**:		Republic of Serbia	
Locality of publication, **LP**:		Vojvodina, Novi Sad	
Publication year, **PY**:		2026.	
Publisher, **PB**:		Author's reprint	
Publication place, **PP**:		Novi Sad, Trg Dositeja Obradovića 6	
Physical description, **PD**: (chapters/pages/ref./tables/pictures/graphs/appendixes)		8/78/19/24/20/0	
Scientific field, **SF**:		IMT - Information technologies	
Scientific discipline, **SD**:		Information systems engineering	
Subject/Key words, **S**/**KW**:		Microfrontends, Web applications	
UC			
Holding data, **HD**:		Library, Faculty of Technical Sciences, Trg Dositeja Obradovića 6, Novi Sad	
Note, **N**:		None	
Abstract, **AB**:		The paper explores microfrontend architecture through three approaches: iframe, Webpack Module Federation, and Single-SPA, implemented on an identical e-commerce application (B-World) using React, Vue.js, and Svelte. Lighthouse measurements showed that Module Federation achieves the best performance (334 KB), iframe provides isolation for legacy systems, and Single-SPA offers the greatest flexibility. The result is a decision matrix for selecting the optimal approach.	
Accepted by the Scientific Board on, **ASB**:			
Defended on, **DE**:			
Defended Board, **DB**:	President:	Andras Anderla, Ph.D.	
	Member:	Marko Arsenovic, Ph.D.	Menthor's sign
	Member, Mentor:	Srdjan Sladojevic, Ph.D.	

	УНИВЕРЗИТЕТ У НОВОМ САДУ • **ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА** 21000 НОВИ САД, Трг Доситеја Обрадовића 6	Датум:
	ЗАДАТАК ЗА ИЗРАДУ ДИПЛОМСКОГ (BACHELOR) РАДА	Лист/Листова: 1/1

(Податке уноси предметни наставник - ментор)

Врста студија:	●Основне академске студије ○Основне струковне студије
Студијски програм:	**Инжењерство информационих система**
Руководилац студијског програма:	**Др Душанка Дакић, доцент**

Студент:	**Младен Бајић**	Број индекса:	**ИТ 46/2018**
Област:	**ИМТ – информационе технологије**		
Ментор:	**Др Срђан Сладојевић, редовни професор**		

НА ОСНОВУ ПОДНЕТЕ ПРИЈАВЕ, ПРИЛОЖЕНЕ ДОКУМЕНТАЦИЈЕ И ОДРЕДБИ СТАТУТА ФАКУЛТЕТА

ИЗДАЈЕ СЕ ЗАДАТАК ЗА ДИПЛОМСКИ (Bachelor) РАД, СА СЛЕДЕЋИМ ЕЛЕМЕНТИМА:

1. проблем – тема рада;
2. начин решавања проблема и начин практичне провере резултата рада, ако је таква провера неопходна;
3. литература

НАСЛОВ ДИПЛОМСКОГ (BACHELOR) РАДА:

Микро фронтенд архитектура: Теорија и примјена у развоју веб апликација

ТЕКСТ ЗАДАТКА:

1. Анализирати постојеће приступе имплементацији микрофронтенд архитектуре и идентификовати кључне карактеристике, предности и ограничења сваког.
2. Пројектовати архитектуру *e-commerce* апликације са три микрофронтенда: *Container*, *Book List* и *Single Book*, уз дефинисање комуникационих образаца.
3. Имплементирати апликацију путем *iframe* приступа са *postMessage* комуникацијом, интегришући *React*, *Vue*.js и *Svelte* у јединствену апликацију уз *pnpm* монорепозиторијум.
4. Имплементирати апликацију коришћењем *Webpack Module Federation* и *Vite* алата, са дељеним зависностима и *React Context* стањем.
5. Имплементирати апликацију у *Single-SPA* оквиру са *Root Config*, *Import Maps*, *Layout Engine* и *RxJS Event Bus* комуникацијом. У свим имплементацијама интегрисати *Google Books API* са функционалностима претраге, филтрирања и приказа детаља.

Руководилац студијског програма:	Ментор рада:
Др Душанка Дакић, доцент	**Др Срђан Сладојевић, редовни професор**

Примерак за: ● - Студента; ○ - Ментора

УНИВЕРЗИТЕТ У НОВОМ САДУ • **ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА**
21000 НОВИ САД, Трг Доситеја Обрадовића 6

ИЗЈАВА О НЕПОСТОЈАЊУ СУКОБА ИНТЕРЕСА

Изјављујем да нисам у сукобу интереса у односу ментор – кандидат и да нисам члан породице (супружник или ванбрачни партнер, родитељ или усвојитељ, дете или усвојеник), повезано лице (крвни сродник ментора/кандидата у правој линији, односно у побочној линији закључно са другим степеном сродства, као ни физичко лице које се према другим основама и околностима може оправдано сматрати интересно повезаним са ментором или кандидатом), односно да нисам зависан/на од ментора/кандидата, да не постоје околности које би могле да утичу на моју непристрасност, нити да стичем било какве користи или погодности за себе или друго лице било позитивним или негативним исходом, као и да немам приватни интерес који утиче, може да утиче или изгледа као да утиче на однос ментор-кандидат.

У Новом Саду, дана _____

Ментор

_____

Кандидат

_____

Садржај

1. Увод....8
2. Микрофронтенд архитектура....10
2.1. Историјски контекст и еволуција....10
2.2. Монолитна фронтенд архитектура....10
2.3. Дефиниција и основни концепти микрофронтенд архитектуре....11
2.4. Предности микрофронтенд архитектуре....12
2.5. Изазови и недостаци микрофронтенд архитектуре....12
2.6. Обрасци интеграције микрофронтенда....12
2.6.1. Интеграција путем iframe елемената....13
2.6.2. Интеграција путем JavaScript радних оквира (Single-SPA)....13
2.6.3. Интеграција путем Module Federation....13
2.7. Критеријуми за одабир приступа....14
2.8. Примјери из индустрије....14
3. Преглед приступа микрофронтенд архитектури....15
3.1. Интеграција путем iframe елемената....15
3.1.1. Принцип рада....15
3.1.2. Комуникација путем postMessage API-ja....15
3.1.3. Предности iframe приступа....16
3.1.4. Недостаци iframe приступа....16
3.1.5. Случајеви употребе....17
3.2. Webpack Module Federation....17
3.2.1. Module Federation Архитектура....17
3.2.2. Module Federation Конфигурација....18
3.2.3. Предности Module Federation приступа....18
3.2.4. Недостаци Module Federation приступа....19
3.2.5. Напредни обрасци....19
3.3. Single-SPA мета радни оквир....20
3.3.1. Концепт апликационе љуске....20
3.3.2. Методе животног циклуса....20
3.3.3. Рутирање и активација....21
3.3.4. Import Maps и дијељење зависности....21
3.3.5. Комуникација између микрофронтенда....21
3.3.6. Предности Single-SPA приступа....22
3.3.7. Недостаци Single-SPA приступа....22
3.4 Компаративна анализа приступа....23
4. Технологије коришћене у имплементацији....24
4.1. Заједничке технологије....24
4.1.1. Vite....24
4.1.2. TypeScript....24
4.1.3. TailwindCSS....24
4.1.4. Google Books API....24
4.2. Технологије iframe приступа....25
4.2.1. React....25
4.2.2. Svelte....25
4.2.3. Vue.js....25

4.2.4. Pnpm радни простор(монорепозиторијум)....26
4.2.5. postMessage API....26
4.3. Технологије Module Federation приступа....26
4.3.1. Webpack Module Federation....26
4.3.2. @originjs/vite-plugin-federation....27
4.3.3. React Context API....27
4.3.4. React Suspense и Lazy учитавање....27
4.4. Технологије Single-SPA приступа....27
4.4.1. Single-SPA радни оквир....28
4.4.2. single-spa-layout....28
4.4.3. Import Maps....28
4.4.4. single-spa-react....28
4.4.5. RxJS (Reactive Extensions for JavaScript)....29
4.5. Упоредни преглед технологија....29
5. Концепт Ријешења....30
5.1. Опис апликације....30
5.1.1 Циљеви апликације....30
5.1.2. Кориснички профил....30
5.2. Функционални захтјеви....31
5.3. Архитектура система....31
5.3.1. Подјела на микрофронтенде....31
5.3.2. Ток података....32
5.3.3. Варијације архитектуре по приступу....32
5.4. Извор података - Google Books API....33
5.4.1. Структура API-ја....33
5.4.2. Обрада података....33
5.5. Кориснички сценарији....34
5.5.1. Сценарио: Претрага и куповина књиге....34
5.5.2. Сценарио: Истраживање каталога....34
5.6. Дизајн корисничког интерфејса....34
5.6.1. Распоред странице....34
5.6.2. Компоненте интерфејса....35
5.6.3. Респонзивни дизајн....35
6. Имплементација....36
6.1. Имплементација iframe приступа....36
6.1.1. Структура пројекта....36
6.1.2. Дијељени модул за комуникацију....38
6.1.3. Container апликација (React)....40
6.1.4. Book List микрофронтенд (Svelte)....42
6.1.5. Single Book микрофронтенд (Vue)....44
6.1.6. Дијаграм комуникације....46
6.2. Имплементација Module Federation приступа....46
6.2.1. Структура пројекта....47
6.2.2. Module Federation Конфигурација....48
6.2.3. Дијељени Глобални Контекст....48
6.2.4. Host апликација са динамичким учитавањем....50

6.2.5. Book List Remote модул..... 51
6.3. Имплементација Single-SPA приступа..... 53
6.3.1. Структура пројекта..... 53
6.3.2 Root Config и Import Maps..... 54
6.3.3. Event Bus - RxJS комуникација..... 55
6.3.4. React микрофронтенд интеграција..... 56
6.3.5. Header микрофронтенд..... 58
6.3.6. Дијаграм RxJS комуникације..... 59
6.4. Завршне напомене о имплементацији..... 60
7. Компаративна Анализа..... 61
7.1. Критеријуми евалуације..... 61
7.2. Анализа перформанси..... 62
7.2.1. Вријеме учитавања..... 62
7.2.2. Величина bundle-a..... 63
7.2.3 Lighthouse оцјене..... 64
7.3. Анализа сложености имплементације..... 64
7.3.1. Конфигурациона сложеност..... 65
7.3.2. Интеграциони код..... 65
7.3.3. Крива учења..... 65
7.4. Искуство програмера..... 67
7.4.1. Процес развоја софтвера..... 67
7.4.2. Могућности за дебаговање..... 67
7.5. Скалабилност..... 68
7.5.1 Скалабилност броја микрофронтенда..... 68
7.5.2. Скалабилност паралелног развоја..... 68
7.6. Одржавање и дугорочна одрживост..... 69
7.6.1. Управљање зависностима..... 69
7.6.2. Изолација грешака..... 69
7.7. Препоруке за избор приступа..... 70
7.7.1. Када изабрати iframe приступ..... 70
7.7.2. Када изабрати Module Federation..... 70
7.7.3. Када изабрати Single-SPA..... 70
7.7.4. Матрица одлучивања..... 71
7.8. Закључак компаративне анализе..... 72
8. Закључак..... 73
8.1. Преглед остварених циљева..... 73
8.2. Кључни налази истраживања..... 73
8.2.1. Налази о iframe приступу..... 73
8.2.2. Налази о Module Federation приступу..... 74
8.2.3. Налази о Single-SPA приступу..... 74
8.3. Доприноси рада..... 74
8.4. Ограничења истраживања..... 75
8.5. Препоруке за будућа истраживања..... 75
8.6 Закључак..... 76
Референце..... 77

Списак Слика

Слика 6.1 - Почетни екран *B-World* апликације
Слика 6.2 - Визуелни приказ монорепозиторијумске структуре *iframe* пројекта
Слика 6.3 - *Container* апликација - *React*
Слика 6.4 - *Book List* микрофронтенд - *Svelte*
Слика 6.5 - *Single Book* микрофронтенд - *Vue*
Слика 6.6 - Дијаграм тока комуникације у *iframe* приступу
Слика 6.7 - Архитектура *Module Federation* са *Host* и *Remote* апликацијама
Слика 6.8 - *Module Federation Host* са учитаним *Remote* модулима
Слика 6.9 - *Book List Remote* са дијељеним контекстом
Слика 6.10 - Архитектура *Single-SPA* са *Root Config* оркестратором
Слика 6.11 - *Single-SPA Book List* са *RxJS* комуникацијом
Слика 6.12 - *Single-SPA* Header са корпом
Слика 6.13 - Дијаграм *RxJS publish-subscribe* комуникације
Слика 6.14 - Упоредни приказ сва три приступа
Слика 7.1 - Графикон упоређења времена учитавања (FCP, LCP, TTI)
Слика 7.2 - Графикон расподјеле величине *bundle-a* по компонентама
Слика 7.3 - Радар графикон *Lighthouse* оцјена за сва три приступа
Слика 7.4 - Дијаграм криве учења током времена за сва три приступа
Слика 7.5 - Приказ поређења алата за дебаговање за сва три приступа
Слика 7.6 - Дијаграм тока одлучивања за избор приступа

Списак Табела

Табела 3.1 - Упоредни преглед приступа микрофронтенд интеграцији
Табела 4.1 - Технологије *iframe* приступа
Табела 4.2 - Технологије *Module Federation* приступа
Табела 4.3 - Структура *Single-SPA* пројекта
Табела 4.4 - Упоредни преглед технологија
Табела 5.1 - Функционални захтјеви апликације
Табела 5.2 - Подјела на микрофронтенде
Табела 5.3 - Структура одговора *Google Books API*-ја
Табела 6.1 - Преглед порука у *iframe* комуникацији
Табела 6.2 - *Module Federation* изложени модули
Табела 6.3 - *RxJS Observable* стања
Табела 7.1 - Преглед критеријума евалуације
Табела 7.2 - Резултати мјерења времена учитавања
Табела 7.3 - Величина *bundle*-а по приступу
Табела 7.4 - *Lighthouse* оцјене
Табела 7.5 - Конфигурациона сложеност
Табела 7.6 - Обим интеграционог кода
Табела 7.7 - Процјена криве учења
Табела 7.8 - Поређење различитих процеса у развоју софтвера
Табела 7.9 - Скалабилност при повећању броја микрофронтенда
Табела 7.10 - Подршка за паралелни развој
Табела 7.11 - Изолација грешака по типу
Табела 7.12 - Матрица одлучивања за избор приступа
Табела 8.1 - Преглед остварених циљева истраживања

Списак Листинга

Листинг 6.1 - Структура *iframe* пројекта
Листинг 6.2 - Валидација поријекла и слање порука
Листинг 6.3 - Дефиниција типова порука
Листинг 6.4 - Container *App.tsx* - рутирање порука
Листинг 6.5 - *Book List App.svelte*
Листинг 6.6 - *Single Book App.vue*
Листинг 6.7 - Структура *Module Federation* пројекта
Листинг 6.8 - *Host vite.config.ts*
Листинг 6.9 - *Remote vite.config.ts* (*book-list-react*)
Листинг 6.10 - *GlobalContext* дефиниција
Листинг 6.11 - *Host App.tsx*
Листинг 6.12 - *Book List Remote App.tsx*
Листинг 6.13 - Структура *Single-SPA* пројекта
Листинг 6.14 - *Import Maps* конфигурација
Листинг 6.15 - Микрофронтед *Layout*
Листинг 6.16 - *Root Config main.ts*
Листинг 6.17 - *Event Bus* имплементација
Листинг 6.18 - *Event Bus* животни циклус и глобално излагање
Листинг 6.19 - *Book List main.ts*
Листинг 6.20 - *Book List App.tsx* са *RxJS* претплатом
Листинг 6.21 - *Header App.tsx*

Скраћенице

API - енг. *Application Programming Interface*
CDN - енг. *Content Delivery Network*
CLS - енг. *Cumulative Layout Shift*
CSS - енг. *Cascading Style Sheets*
DOM - енг. *Document Object Model*
DX - енг. *Developer Experience*
ES - енг. *ECMAScript*
FCP - енг. *First Contentful Paint*
HMR - енг. *Hot Module Replacement*
HTML - енг. *HyperText Markup Language*
HTTP - енг. *HyperText Transfer Protocol*
JS - енг. *JavaScript*
JSON - енг. *JavaScript Object Notation*
KB - енг. *Kilobyte*
LCP - енг. *Largest Contentful Paint*
LOC - енг. *Lines of Code*
MF - енг. *Module Federation*
NPM - енг. *Node Package Manager*
RxJS - енг. *Reactive Extensions for JavaScript*
SEO - енг. *Search Engine Optimization*
SPA - енг. *Single Page Application*
SSG - енг. *Static Site Generation*
SSR - енг. *Server-Side Rendering*
TBT - енг. *Total Blocking Time*
TTI - енг. *Time to Interactive*
UI - енг. *User Interface*
URL - енг. *Uniform Resource Locator*
UX - енг. *User Experience*

1. Увод

У савременом софтверском развоју, повећање сложености софтверских производа довело је до потребе за развојем архитектура и парадигми које омогућавају лакше скалирање тимова, брже достављање функционалности и бољу модуларност самог кода. Један од таквих приступа, који се посљедњих неколико година наметнуо као изузетно релевантан, јесте микрофронтенд архитектура.

Традиционалне фронтенд апликације често се развијају као монолитне структуре, односно један велики фронтенд пројекат који покрива све функционалности апликације. Овакав приступ може одлично функционисати у малим тимовима, али како се пројекат развија и како број програмера расте, јављају се потешкоће као што су: спорија комуникација, нејасна расподјела одговорности за одржавање кода, смањена флексибилност у избору технологија и отежан развој нових функционалности. Логички независни дијелови апликације у монолитној структури често имају непотребне зависности. Микрофронтенд архитектура разбија монолит тако што дијели фронтенд на мање, независне модуле (тзв. микрофронтенде) које развијају, тестирају и одржавају засебни тимови.

Микрофронтенд архитектура омогућава вертикалну подјелу у развоју софтверских производа. Сваки тим посједује одговорност за развој логички одвојеног дијела апликације, укључујући фронтенд, бекенд и све слојеве које тим одлучи да користи у развоју софтверског производа. На овај начин, тимови могу независно развијати функционалности, користити различите технологије када је то потребно и имати веома брзе итерације у развоју, што иде руку под руку са агилним принципима развоја софтвера. Овај приступ се нарочито показао успјешним у средњим и великим организацијама, гдје агилни тимови могу ефикасно радити на сопственим микроапликацијама, одржавајући конзистентност производа као цјелине.

У литератури се наводе кључне предности микрофронтенд архитектуре:

- **Оптимизација развоја функционалности**: тимови имају све вјештине и ресурсе за независан развој и испоруку дијела (функционалности) софтверског производа.
- **Независност у одабиру технологија**: тимови могу независно мијењати избор технологија које користе у развоју, без утицаја других тимова.
- **Фокус на корисничке потребе**: тимови се организују према логичким цјелинама и корисничким потребама, што им омогућава бржу и прецизнију реакцију на потребе тржишта.

Наравно, микрофронтенд архитектура доноси и одређене изазове: редундантност кода, повећана сложеност инфраструктуре, конзистентност података и перформанси, као и одржавање јасне комуникације између тимова.

Циљ овог рада је да пружи преглед микрофронтенд архитектуре, укључујући основне идеје, предности и мане, основне обрасце и примјену, уз посебан осврт на практичну реализацију кроз три пројекта развијена у оквиру овог рада:

1. Микрофронтенд архитектура коришћењем *iframe* елемената
2. Микрофронтенд архитектура коришћењем *Single-SPA* радног оквира
3. Микрофронтенд архитектура коришћењем *Module Federation* алата

Основна хипотеза овог рада јесте да различити приступи имплементацији микрофронтенд архитектуре имају значајно различите карактеристике у погледу сложености имплементације, начина комуникације између модула, могућности дијељења кода и погодности за одређене типове пројеката. Кроз практичну имплементацију исте апликације употребом три различита приступа, рад настоји да идентификује предности и недостатке сваког од њих, те да пружи смјернице за одабир одговарајућег приступа у зависности од конкретних захтјева пројекта.

Рад је структуиран на сљедећи начин. У другом поглављу представљене су теоријске основе микрофронтенд архитектуре, укључујући историјски контекст, поређење са монолитном архитектуром и преглед основних образаца интеграције. Треће поглавље детаљно описује различите приступе имплементацији микрофронтенд архитектуре: *iframe* интеграцију, *Module Federation* и *Single-SPA* радни оквир. Четврто поглавље представља технологије коришћене у имплементацији практичног дијела рада. Пето поглавље приказује концепт рјешења и функционалне захтјеве апликације за продају књига која служи као основа за сва три приступа. Шесто поглавље представља срж практичног дијела рада и детаљно описује имплементацију сваког од три приступа, укључујући структуру пројеката, начин комуникације између микрофронтенда и кључне техничке одлуке. Седмо поглавље садржи компаративну анализу имплементираних рјешења према унапријед дефинисаним критеријумима. Осмо поглавље доноси закључак, сумира добијене резултате и предлаже правце будућег рада и истраживања.

2. Микрофронтенд архитектура

2.1. Историјски контекст и еволуција

Развој веб апликација прошао је кроз неколико значајних фаза еволуције. У раним данима веба, веб странице су биле статичне и једноставне, са минималном интерактивношћу. Увођењем *JavaScript* језика 1995. године омогућена је динамичка манипулација садржајем на клијентској страни, што је представљало прекретницу у развоју веб технологија.

Појава *Ajax* технологије средином 2000-их година омогућила је асинхрону комуникацију са сервером без потребе за поновним учитавањем странице. Ово је отворило пут ка развоју богатих интернет апликација (*Rich Internet Applications*) и касније једностраничних апликација (*Single Page Applications*, SPA). Радни оквири попут *AngularJS* (2010), *React* (2013) и *Vue.js* (2014) донијели су структуру и организацију у фронтенд развој, омогућавајући креирање сложених корисничких интерфејса.

Паралелно са овим развојем, на бекенд страни се дешавала слична трансформација. Монолитне серверске апликације постепено су уступале мјесто микросервисној архитектури, коју је популаризовала компанија *Netflix* почетком 2010-их година. Микросервисна архитектура омогућила је подјелу бекенд система на мање, независне сервисе који комуницирају путем мрежних протокола.

Међутим, док је бекенд прошао кроз ову трансформацију, фронтенд је углавном остајао монолитан. Овај дисбаланс постао је очигледан у великим организацијама гдје су тимови користили микросервисе на бекенду, али су се и даље борили са једном великом фронтенд апликацијом. Термин „микрофронтенд" први пут је употријебио Michael Geers 2016. године, а исте године појавио се и у *ThoughtWorks Technology Radar* публикацији, означавајући формално признање овог архитектонског приступа.

2.2. Монолитна фронтенд архитектура

Монолитна фронтенд архитектура подразумијева развој цјелокупног корисничког интерфејса као једне, обједињене кодне базе. У овом моделу, сви дијелови апликације дијеле исти репозиторијум, исте зависности, исти процес изградње и исти процес постављања на продукциону околину.

Монолитни приступ има одређене предности које га чине погодним за одређене сценарије:

- **Једноставност развоја**: Једна кодна база олакшава навигацију и разумијевање цјелокупног система, посебно за мање тимове.
- **Конзистентност**: Дијељење кода и стилова је директно, што олакшава одржавање визуелне и функционалне конзистентности.
- **Једноставнија инфраструктура**: Један процес изградње и достављања смањује оперативну сложеност.
- **Лакше тестирање**: Интеграционо тестирање је директније када је сав код у једном мјесту.

Међутим, како апликација расте и број програмера се повећава, монолитна архитектура почиње да испољава озбиљне недостатке:

- **Проблеми са скалирањем тимова**: Када више тимова ради на истој кодној бази, долази до честих конфликата при спајању кода, нејасне подјеле одговорности и потребе за интензивном координацијом.
- **Спорост развојног циклуса**: Вријеме изградње (*build time*) расте са величином кодне базе. Мала промјена захтијева поновну изградњу цијеле апликације.
- **Технолошка ограничења**: Промјена технолошког стека (*tech stack*) или надоградња верзије радног оквира захтијева модификацију цјелокупне апликације, што представља значајан ризик.
- **Отежана испорука**: Свака промјена, без обзира на величину, захтијева испоруку цјелокупне апликације, повећавајући ризик и успоравајући циклус испоруке.
- **Акумулација техничког дуга**: Чврста повезаност компоненти отежава рефакторисање и доводи до постепеног нагомилавања техничког дуга.

2.3. Дефиниција и основни концепти микрофронтенд архитектуре

Микрофронтенд архитектура представља архитектонски образац у развоју веб апликација гдје се независно развијени фронтенд модули композитно уклапају у већу цјелину. Овај приступ је аналоган микросервисној архитектури, али примијењен на клијентску страну једностраничних апликација (SPA) писаних у *JavaScript* језику.

Према дефиницији са веб портала *micro-frontends.org*, који одржава Michael Geers, микрофронтенд архитектура се заснива на идеји да се о веб сајту или веб апликацији размишља као о композицији функционалности које су у власништву независних тимова. Сваки тим има јасно дефинисану област пословања или мисију за коју је специјализован, а тим је крос-функционалан и развија своје функционалности од базе података до корисничког интерфејса.

Кључни концепти микрофронтенд архитектуре укључују:

- **Независност тимова**: Сваки тим посједује пуну аутономију над својим микрофронтендом. Тим доноси одлуке о технологијама, архитектури и процесима развоја без потребе за координацијом са другим тимовима.
- **Независне испоруке**: Свакі микрофронтенд има сопствени репозиторијум, процес изградње, тестирања и постављања. Тимови могу постављати нове верзије својих микрофронтенда без утицаја на остатак система.
- **Технолошка агностичност**: Различити микрофронтенди могу користити различите технологије и радне оквире. Ово омогућава тимовима да бирају најбоље алате за своје специфичне потребе и олакшава постепену миграцију застарјелих система.
- **Изолација грешака**: Проблем у једном микрофронтенду не би требало да утиче на функционисање осталих дијелова апликације, повећавајући укупну отпорност система.
- **Вертикална подјела**: За разлику од хоризонталне подјеле (гдје су фронтенд и бекенд одвојени), микрофронтенд архитектура промовише вертикалну подјелу гдје један тим посједује цјелокупну функционалност од корисничког интерфејса до базе података.

2.4. Предности микрофронтенд архитектуре

Микрофронтенд архитектура доноси бројне предности које су посебно изражене у великим организацијама са више развојних тимова:

- **Независан развој и постављање**: Тимови могу развијати, тестирати и постављати своје микрофронтенде без координације са другим тимовима. Ово значајно убрзава циклус испоруке и смањује ризик повезан са постављањем.
- **Инкрементална надоградња**: Застарјели дијелови апликације могу се постепено замјењивати новим имплементацијама. Ово омогућава миграцију на нове технологије без потребе за потпуним преписивањем апликације.
- **Мање и једноставније кодне базе**: Свака кодна база микрофронтенда је мања и једноставнија за разумијевање. Нови чланови тима могу брже постати продуктивни јер не морају да савладају цјелокупну апликацију.
- **Скалирање тимова**: Организација може расти додавањем нових тимова који преузимају одговорност за нове микрофронтенде, без повећања сложености постојећих тимова.
- **Експериментисање и иновације**: Тимови могу експериментисати са новим технологијама и приступима у изолованом окружењу, без ризика за цјелокупан систем.
- **Брже вријеме изградње**: Мање кодне базе значе краће вријеме достављања функционалности, што побољшава продуктивност програмера и убрзава повратне информације.

2.5. Изазови и недостаци микрофронтенд архитектуре

Упркос бројним предностима, микрофронтенд архитектура доноси и значајне изазове које треба пажљиво размотрити:

- **Повећана оперативна сложеност**: Више репозиторијума, више процеса изградње и достављања, више окружења за постављање. Све ово захтијева робуснију инфраструктуру и аутоматизацију.
- **Редундантност кода и зависности**: Различити микрофронтенди могу дуплицирати заједничке зависности, повећавајући укупну величину апликације коју корисник мора да преузме.
- **Конзистентност корисничког искуства**: Одржавање визуелне и функционалне конзистентности захтијева додатни напор у виду дијељених библиотека компоненти и система дизајна.
- **Комуникација између микрофронтенда**: Ефикасан пренос података и стања између независних микрофронтенда захтијева пажљиво планирање и имплементацију.
- **Тестирање интеграције**: Тестирање интеракције између различитих микрофронтенда може бити сложеније него у монолитној апликацији.
- **Перформансе**: Учитавање више независних *bundle*-ова може утицати на почетно вријеме учитавања апликације ако се не имплементира пажљиво.

2.6. Обрасци интеграције микрофронтенда

Постоји неколико приступа за интеграцију микрофронтенда у јединствену апликацију. Сваки приступ има своје предности и недостатке, а избор зависи од специфичних захтјева пројекта.

2.6.1. Интеграција путем *iframe* елемената

Iframe (*inline Frame*) је HTML елемент који омогућава уградњу једног HTML документа унутар другог. Овај приступ је један од најстаријих начина за композицију веб садржаја и нуди највиши степен изолације између микрофронтенда.

Предности *iframe* приступа укључују: потпуну изолацију *JavaScript* и CSS кода, једноставност имплементације и могућност интеграције апликација развијених у било којој технологији. Изолација коју пружа *iframe* значи да грешка у једном микрофронтенду не може директно утицати на друге дијелове апликације.

Комуникација између *iframe* елемената и главне апликације остварује се путем *window.postMessage()* API-ја, који омогућава сигурну размјену порука између различитих извора (*cross-origin communication*). Овај механизам омогућава пренос података и координацију између изолованих контекста.

Недостаци *iframe* приступа укључују: ограничења у интеграцији корисничког искуства, потенцијалне проблеме са перформансама због учитавања засебних контекста за сваки *iframe*, сложеност имплементације респонзивног дизајна и изазове са дубоким повезивањем (*deep linking*).

2.6.2. Интеграција путем *JavaScript* радних оквира (*Single-SPA*)

Single-SPA је *JavaScript* радни оквир који омогућава коегзистенцију више микрофронтенда, потенцијално развијених у различитим технологијама, унутар исте апликације. Радни оквир управља животним циклусом сваког микрофронтенда, омогућавајући динамичко учитавање и уклањање на основу конфигурације рутирања.

Single-SPA уводи концепт животног циклуса микрофронтенда који се састоји од следећих фаза:

- ***Bootstrap***: Иницијализација микрофронтенда, позива се једном приликом првог учитавања.
- ***Mount***: Рендеровање микрофронтенда у документ (DOM), позива се сваки пут када микрофронтенд треба да буде приказан.
- ***Unmount***: Уклањање микрофронтенда из документа (DOM), позива се када микрофронтенд више није активан.

Предности *Single-SPA* укључују: подршку за више радних оквира (*React*, *Vue*, *Angular*, *Svelte*), централизовано управљање рутирањем и флексибилност у организацији микрофронтенда. Овај приступ је посебно користан у сценаријима миграције гдје постојеће апликације треба постепено замијенити новим имплементацијама.

Недостаци укључују: сложенију почетну конфигурацију, потребу за прилагођавањем постојећих апликација и изазове у дијељењу стања између микрофронтенда.

2.6.3. Интеграција путем *Module Federation*

Module Federation је функционалност уведена у *Webpack 5* која омогућава *JavaScript* апликацијама да динамички учитавају код из других апликација током извршавања. Овај приступ је представљен 2020. године и сматра се значајним напретком у архитектури *JavaScript* апликација.

Module Federation уводи концепте домаћина (*host*) и удаљеног модула (*remote*):

- ***Host* (домаћин)**: Апликација која учитава и користи модуле из других апликација.
- ***Remote* (удаљени модул)**: Апликација која излаже своје модуле за коришћење од стране других апликација.
- ***Shared* (дијељене зависности)**: Зависности које се дијеле између апликација како би се избјегло дуплицирање кода.

Предности *Module Federation* укључују: могућност дијељења кода током извршавања без потребе за поновном изградњом, ефикасно управљање зависностима и дубоку интеграцију компоненти. Дијељене зависности омогућавају да се библиотеке попут *React*-а учитају само једном, смањујући укупну величину апликације.

Недостаци укључују: везаност за *Webpack* екосистем (иако постоје алтернативе за *Vite* и друге алате), сложеност почетне конфигурације и потенцијалне проблеме са компатибилношћу верзија дијељених зависности.

2.7. Критеријуми за одабир приступа

Одабир одговарајућег приступа за имплементацију микрофронтенд архитектуре зависи од више фактора:

- **Потреба за изолацијом**: Ако је потребна потпуна изолација између микрофронтенда, *iframe* приступ може бити најбољи избор.
- **Технолошка разноликост**: Ако различити тимови користе различите радне оквире, *Single-SPA* пружа добру подршку за овакве сценарије.
- **Дијељење кода**: Ако је потребно ефикасно дијељење компоненти и библиотека, *Module Federation* нуди најнапредније могућности.
- **Постојећа инфраструктура**: Ако је тим већ упознат са *Webpack*-ом, *Module Federation* може бити природан избор.
- **Сложеност миграције**: За постепену миграцију постојећих апликација, *iframe* приступ захтијева најмање промјена у постојећем коду.

2.8. Примјери из индустрије

Бројне велике компаније успјешно примјењују микрофронтенд архитектуру:

Spotify користи микрофронтенде за управљање различитим дијеловима своје веб и десктоп апликације. Различити тимови раде на функционалностима попут управљања плејлистама, претраге и корисничког профила.

IKEA је имплементирала микрофронтенде комбинујући *Edge Side Includes* (ESI) са *Client Side Includes* (CSI), омогућавајући независан развој и постављање различитих дијелова своје е-трговинске платформе.

Spotify*, *Netflix*, *Amazon*, *PayPal* и *eBay су међу компанијама које су препознале вриједност овог архитектонског приступа за скалирање својих фронтенд тимова и убрзање испоруке нових функционалности.

3. Преглед приступа микрофронтенд архитектури

Микрофронтенд архитектура може се реализовати кроз различите техничке приступе, од којих сваки нуди специфичан баланс између изолације, перформанси, сложености имплементације и флексибилности. У овом поглављу детаљно су представљена три приступа која се користе у практичном дијелу рада: интеграција путем *iframe* елемената, *Webpack Module Federation*, и *Single-SPA* мета радни оквир. Разумијевање механизама сваког приступа неопходно је за доношење информисане одлуке о томе који приступ најбоље одговара конкретним захтјевима пројекта.

3.1. Интеграција путем *iframe* елемената

Iframe (*inline frame*) елемент је HTML елемент који омогућава уграђивање једног HTML документа унутар другог. Ова техника представља најстарији и најједноставнији начин композиције независних веб апликација. Иако се *iframe* често сматра застарјелом технологијом, у контексту микрофронтенда нуди јединствене предности које га чине релевантним избором за одређене сценарије.

3.1.1. Принцип рада

Када веб прегледач наиђе на *iframe* елемент, он креира потпуно нови контекст прегледања (*browsing context*) за уграђени документ. Овај контекст укључује сопствени *window* објекат, *document* објекат, историју навигације, и изоловано *JavaScript* вријеме извршавања. Из перспективе прегледача, сваки *iframe* је еквивалентан отварању нове картице (*tab*), с тим да се садржај визуелно приказује унутар родитељске странице.

У микрофронтенд архитектури, главна (*container*) апликација служи као љуска која дефинише распоред странице и позиционира *iframe* елементе на одговарајућа мјеста. Сваки *iframe* учитава независну апликацију са сопственог сервера. Ове апликације могу бити развијене коришћењем потпуно различитих технологија, верзија библиотека, па чак и различитих тимова са независним циклусима испоруке.

Структура типичне *iframe* интеграције подразумијева да главна апликација генерише HTML документ са *iframe* елементима чији *src* атрибути показују на URL адресе појединачних микрофронтенда. Примјер таквог *markup*-а је:

<iframe src="https://team-a.example.com/component" />.

Прегледач затим паралелно учитава све уграђене документе, а сваки од њих пролази кроз комплетан процес парсирања, извршавања скрипти и рендеровања, независно од осталих.

3.1.2. Комуникација путем *postMessage* API-ја

Изолација коју пружа *iframe* елемент значи да уграђени документ не може директно приступити структури документа (DOM) или *JavaScript* варијаблама родитељског документа, и обрнуто. Ово понашање, познато као правило истог поријекла (*Same-Origin Policy)*, представља фундаментални безбједносни механизам веба. Међутим, микрофронтенди често морају размјењивати податке, на примјер када корисник кликне на производ у листи (један микрофронтенд) и очекује да се прикажу детаљи (други микрофронтенд).

Window.postMessage() API омогућава сигурну комуникацију између докумената различитог поријекла. Овај механизам функционише на принципу размјене порука: пошиљалац позива *postMessage()* метод на *window* објекту примаоца, прослеђујући податке и циљно

поријекло. Прималац ослушкује *'message'* догађај и обрађује примљене податке након валидације поријекла пошиљаоца.

Синтакса слања поруке је: *targetWindow.postMessage(data, targetOrigin)*, гдје је *targetWindow* референца на *window* објекат циљног *iframe*-а (доступна преко *contentWindow* својства), *data* је објекат који се шаље (аутоматски серијализован), а *targetOrigin* је URL поријекла примаоца или '*' за било које поријекло (не препоручује се из безбједносних разлога).

На страни примаоца, обрада изгледа овако:

window.addEventListener('message', function(event) { if (event.origin !== 'https://trusted-origin.com') return; / обрада event.data */ }).*

Провјера *event.origin* је критична за безбједност јер спречава злонамјерне скрипте да шаљу лажне поруке.

3.1.3. Предности *iframe* приступа

Потпуна изолација представља примарну предност *iframe* приступа. *JavaScript* код унутар *iframe*-а не може случајно или намјерно утицати на родитељски документ или друге *iframe*-ове. Ово елиминише цијелу класу грешака везаних за конфликте глобалних варијабли, колизије CSS селектора, или неочекиване интеракције између библиотека. Geers (2020) истиче да је ова изолација посебно вриједна при интеграцији старих (*legacy*) система или компоненти трећих страна код којих немамо контролу над квалитетом кода.

Технолошка агностичност је друга значајна предност. Пошто сваки *iframe* представља независан документ, не постоје ограничења у погледу технологије коју појединачни микрофронтенд користи. Један тим може користити *React 18*, други *Angular 15*, а трећи *Vue 3*, без икаквих конфликата. Ово омогућава постепену миграцију старих апликација и даје тимовима слободу одабира најбољих алата за њихове специфичне потребе.

Независна испорука је поједностављена јер свака апликација има сопствени процес постављање у продукцију(*deployment*). Ажурирање једног микрофронтенда не захтијева поновну изградњу(*rebuild*) или поновну испоруку(*redeploy*) осталих. Ово драматично смањује ризик приликом испоруке нових верзија и омогућава чешће, мање ризичне испоруке.

Робустност система се повећава јер грешка у једном *iframe*-у не може срушити цијелу апликацију. Ако један микрофронтенд доживи *JavaScript* грешку, остатак странице наставља функционисати нормално. Главна (*container)* апликација може имплементирати резервни приказ(*fallback)* који приказују уколико се неки *iframe* не успије учитати.

3.1.4. Недостаци *iframe* приступа

Перформансе представљају најзначајнији недостатак *iframe* интеграције. Сваки *iframe* креира потпуно нови контекст прегледања, што подразумијева алокацију додатне меморије, креирање нових *JavaScript engine instance*-а, и паралелно учитавање свих потребних ресурса. Прегледач мора обрадити комплетан HTML/CSS/JS за сваки *iframe*, што значајно повећава укупно вријеме учитавања странице и потрошњу ресурса.

Редундантност ресурса је директна посљедица изолације. Ако три микрофронтенда користе *React*, свака ће учитати сопствену копију *React* библиотеке. За *React* библиотеку величине око *40KB* (*gzipped*), ово значи *120KB* само за радни оквир, умјесто потенцијалних *40KB* при дијељеном учитавању. Ова редундантност се мултипликује са сваком додатном библиотеком коју микрофронтенди дијеле.

Проблеми са респонзивним дизајном произилазе из чињенице да *iframe* елементи имају фиксне димензије. Када се садржај унутар *iframe*-а динамички мијења (нпр. проширивање елемента), *iframe* не може аутоматски прилагодити своју висину. Рјешавање овог проблема захтијева додатну комуникацију између *iframe*-а и родитељског документа путем *postMessage*-а, гдје *iframe* јавља своју нову висину, а родитељ ажурира стил(*style)* атрибут *iframe* елемента.

Deep linking и дијељење URL-ова је отежано. URL у адресној траци прегледача одражава само адресу главне(container) апликације, не и стање унутар појединачних *iframe*-ова. Ако корисник жели подијелити линк на специфичну страницу унутар микрофронтенда, потребно је имплементирати механизам синхронизације стања са URL-ом родитељске апликације.

SEO импликације су негативне јер претраживачи традиционално нису индексирали садржај унутар *iframe*-ова као дио родитељске странице. Иако су се модерни претраживачи побољшали у овом погледу, *iframe* садржај и даље може имати слабију видљивост у резултатима претраге у односу на директно уграђени садржај.

Приступачност (*accessibility*) може бити компромитована. Читачи екрана (*screen readers*) и други асистивни алати могу имати потешкоћа са навигацијом кроз вишеструке *iframe*-ове. Фокус менаџмент између *iframe*-ова захтијева пажљиву имплементацију, а навигација тастатуром може бити збуњујућа за кориснике.

3.1.5. Случајеви употребе

Iframe приступ је најпогоднији у сценаријима гдје је изолација примарни захтјев. Интеграција старих (*legacy*) система који користе застарјеле библиотеке или имају проблематичан CSS је типичан примјер. Уграђивање компоненти трећих страна, попут payment форми или *chat widget*-а, такође је идеалан случај употребе. Системи гдје различити тимови имају радикално различите технолошке стекове и процесе испоруке могу имати користи од изолације коју *iframe* пружа.

Iframe није препоручен избор за апликације гдје су перформансе и SEO критични фактори, за апликације са интензивном интеракцијом између компоненти, или за мобилне апликације гдје је меморија ограничена.

3.2. *Webpack Module Federation*

Module Federation је функционалност уведена у *Webpack 5* (октобар 2020) која омогућава динамичко дијељење кода између независно изграђених *JavaScript* апликација током времена извршавања (*runtime*). За разлику од традиционалних приступа дијељењу кода који захтијевају интеграцију током времена изградње (*buildtime*), *Module Federation* омогућава апликацијама да учитавају модуле једна од друге директно у прегледачу, без потребе за централним *build* системом.

3.2.1. *Module Federation* Архитектура

Module Federation уводи концепте *Host* и *Remote* апликација. *Host* апликација је она која учитава и конзумира модуле од других апликација. *Remote* апликација је она која излаже (*expose*) своје модуле за конзумирање од стране других. Важно је напоменути да иста апликација може истовремено бити и *Host* и *Remote*, учитавајући модуле од неких апликација док истовремено излаже своје модуле другима.

Централни елемент *Module Federation* архитектуре је *remoteEntry.js* датотека. Ова датотека, генерисана током времена изградње сваке Remote апликације, садржи манифест свих изложених модула и њихових зависности. Када *Host* апликација жели учитати модул

од *Remote*-а, она прво преузима *remoteEntry.js*, затим на основу манифеста одређује које *chunk*-ове треба учитати, и коначно динамички преузима и извршава потребан код.

Механизам дијељена зависности омогућава да се заједничке библиотеке учитају само једном. Умјесто да свака апликација учита сопствену копију *React*-а, *Module Federation* координира учитавање тако да се *React* учита једном и дијели између свих апликација. Ово се постиже конфигурисањем *shared* низа у *Module Federation* додатаку, гдје се наводе библиотеке које треба дијелити, опционо са семантичким ограничењима верзија.

3.2.2. *Module Federation* Конфигурација

Конфигурација *Module Federation*-а врши се у *Webpack* конфигурационој датотеци путем *ModuleFederationPlugin-a*. За *Remote* апликацију, кључни параметри укључују име који јединствено идентификује апликацију, путању која одређује име генерисане *remoteEntry* датотеке, *exposes* објекат који мапира јавне путање на локалне модуле (нпр. *{'./Button': './src/components/Button'}*), и *shared* низ или објекат који дефинише дијељене зависности.

За *Host* апликацију, конфигурација укључује *remotes* објекат који мапира логичка имена *Remote* апликација на URL-ове њихових *remoteEntry* датотека (нпр. {*remoteApp: 'remoteApp@http://localhost:3001/remoteEntry.js'*}). Када *Host* апликација импортује модул путем *import('remoteApp/Button'), Webpack* извршава овај захтјев коришћењем *remotes* конфигурације, преузима одговарајући код са *Remote* сервера, и враћа модул као да је локални.

Управљање верзијама дијељених зависности је критичан аспект конфигурације. *Module Federation* подржава неколико стратегија: *singleton* осигурава да се учита само једна верзија библиотеке (корисно за *React* који не толерише вишеструке *instance*-е), *requiredVersion* специфицира минималну прихватљиву верзију, а *strictVersion* узрокује грешку ако верзије нису компатибилне. Примјер конфигурације: *shared*: *{ react: { singleton: true, requiredVersion: '^18.0.0' } }*.

3.2.3. Предности *Module Federation* приступа

Ефикасно дијељење зависности елиминише редундантно учитавање библиотека. У типичном сценарију са три *React* микрофронтенда, *React* се учитава само једном, што може уштедјети стотине килобајта пропусног опсега и значајно побољшати вријеме учитавања. Ово је посебно значајно за мобилне кориснике са ограниченим плановима интернета или споријим интернетом.

Интеграција у времену извршавања без ограничења времена изградње (*buildtime*) значи да тимови могу независно развијати, тестирати и деплојати своје апликације. Када *Remote* тим објави нову верзију свог микрофронтенда, *Host* апликације аутоматски добијају ажурирану верзију при сљедећем учитавању странице, без потребе за поновном изградњом(*rebuild*-ом) или поновном публикацијом (*redeploy*-ом) *Host* апликације. Ово омогућава истински независне циклусе испоруке.

Дубока интеграција између микрофронтенда је могућа јер се дијели исто *JavaScript* вријеме извршавања. Микрофронтенди могу дијелити *React* контекст, глобално стање, или чак инстанце сервиса. Ово омогућава ниво интеграције који није могућ са *iframe* приступом, гдје је свака апликација у потпуно изолованом окружењу.

Сигурност типова*(type safety)* може се постићи дијељењем *TypeScript* типова између апликација. *Remote* апликација може експортовати *.d.ts* датотеке које *Host* конзумира, пружајући провјеру типова у моменту компајлирања и IDE подршку за аутоматско допуњавање, чак и за динамички учитане модуле.

Lazy loading је природно подржан. Модули се учитавају тек када су потребни, смањујући иницијално вријеме учитавања. *Webpack* аутоматски генерише одвојене *chunk*-ове за сваки изложени модул, а прегледач их преузима тек када су потребни (*on-demand*).

3.2.4. Недостаци *Module Federation* приступа

Иако *Module Federation* представља напредан приступ микрофронтенд архитектури, постоје одређени недостаци и изазови које треба узети у обзир приликом одабира ове технологије.

Зависност од *build tool* екосистема је еволуирала током времена. Оригинална имплементација *Module Federation*-а била је чврсто везана за *Webpack 5*, што је представљало значајно ограничење. Међутим, развојем *Module Federation 2.0* (*@module-federation/enhanced*) и зрелошћу друштвених додатака(*community plugin*) за друге *bundler*-е, ова ситуација се значајно побољшала. У овом раду коришћен је @originjs/vite-plugin-federation који је показао стабилност и функционалност упоредиву са оригиналном *Webpack* имплементацијом. *Rspack*, модерна алтернатива *Webpack*-у написана у *Rust*-у, садржи нативну подршку за Module Federation. Ипак, важно је напоменути да различите имплементације могу имати мање разлике у функционалности или конфигурационим опцијама.

Ограничена технолошка разноликост произилази из начина на који функционише дијељење зависности. Иако је теоретски могуће комбиновати *React* и *Vue* микрофронтенде, губе се примарне предности *Module Federation*-а јер ови радни оквири немају заједничке зависности вриједне дијељења. У пракси, *Module Federation* најбоље функционише када сви микрофронтенди користе исти радни оквир. Ово га чини мање флексибилним од *iframe* или *Single-SPA* приступа када је потребна подршка за хетерогене технолошке стекове. За организације које желе да различити тимови користе различите радне оквире, други приступи могу бити погоднији.

Конфликти верзија могу бити проблематични. Ако један микрофронтенд захтијева *React 17* а други *React 18*, и оба су конфигурисана као *singleton*, долази до конфликта. *Webpack* ће покушати да ријеши конфликт учитавањем новије верзије, али ово може довести до неочекиваног понашања ако старији микрофронтенд није компатибилан са новијом верзијом. Рјешавање захтијева координацију између тимова за синхронизовану надоградњу, или одустајање од *singleton* конфигурације уз прихватање дуплирања библиотека. *requiredVersion* опција пружа одређену контролу, али захтијева дисциплиновано управљање зависностима.

Сложеност дебаговања расте јер се грешке могу десити у динамички учитаном коду са удаљеног сервера. Трагови грешака могу бити мање информативни, посебно када грешка потиче из *Remote* модула. *Source* мапе захтијевају додатну конфигурацију за правилан рад у радном окружењу, а дебаговање може бити отежан ако *Remote* сервер не емитује *source* мапе или ако постоје проблеми са CORS политикама. *Browser DevTools* екстензије за *Module Federation* могу помоћи, али дебаговање је генерално сложеније него у монолитној апликацији.

3.2.5. Напредни обрасци

Двосмјерно(*Bidirectional)* дијељење омогућава да исте апликације буду истовремено *Host* и *Remote*. *Host* апликација може изложити глобални контекст који *Remote* апликације конзумирају за приступ глобалном стању. Истовремено, *Remote* апликације излажу своје UI компоненте које *Host* рендерује. Овај образац омогућава висок степен интеграције уз задржавање модуларности.

Dynamic remotes омогућавају одређивање URL-а *Remote* апликације током времена извршавања умјесто времена изградње. Ово је корисно за вишеклијент(*multi-tenant)* апликације гдје различити корисници могу имати различите конфигурације микрофронтенда, или за *A/B* тестирање различитих верзија микрофронтенда.

3.3. *Single-SPA* мета радни оквир

Single-SPA је *JavaScript meta* радни оквир специјално дизајниран за изградњу микрофронтенд апликација. За разлику од *Module Federation* који се фокусира на дијељење модула, *Single-SPA* се фокусира на оркестрацију животног циклуса (*lifecycle*) независних *JavaScript* апликација унутар јединственог контејнера. Назив потиче од идеје да се вишеструке *Single-Page Applications* (SPA) комбинују у једну кохерентну корисничку апликацију.

3.3.1. Концепт апликационе љуске

Апликациона љуска (или *app shell*) је централна апликација која служи као оркестратор за све микрофронтенде. Geers (2020) описује *app shell* као "тек толико апликације колико је потребно за координацију осталих". У *Single-SPA* терминологији, ова улога припада основној конфигурационој(*root-config*) апликацији.

Root-config апликација је одговорна за неколико критичних функција. Прво, она региструје све микрофронтенде и дефинише услове под којима сваки од њих треба бити активан (обично базирано на URL путањи). Друго, она управља учитавањем *JavaScript* кода сваког микрофронтенда када постане активан. Треће, она координира животни циклус микрофронтенда, позивајући одговарајуће функције животног циклуса (*bootstrap*, *mount*, *unmount*) у правом тренутку.

Важан принцип при дизајнирању *root-config* апликације је да она треба садржавати минималну количину пословне логике. Функционалности попут аутентификације, глобалне навигације, или управљање грешкама могу бити дио *app shell*-а, али треба избјегавати имплементацију доменске логике. Ово осигурава да *app shell* остане стабилан и ријетко захтијева промјене.

3.3.2. Методе животног циклуса

Single-SPA дефинише стандардизовани интерфејс који сваки микрофронтенд мора имплементирати. Овај интерфејс се састоји од три асинхроне функције: *bootstrap*, *mount* и *unmount*. Свака од ових функција враћа *Promise*, омогућавајући асинхроне операције као што су учитавање података или иницијализација библиотека.

Bootstrap функција се позива само једном, када се микрофронтенд први пут активира. Ова функција је намијењена за једнократну иницијализацију, као што је учитавање конфигурације или успостављање веза са *backend* сервисима. Важно је напоменути да *bootstrap* није мјесто за рендеровање UI елемената, јер се позива прије него што је DOM елемент за *mounting* доступан.

Mount функција се позива сваки пут када микрофронтенд постаје активан (нпр. када корисник навигира на одговарајућу руту). У овој функцији микрофронтенд треба рендеровати свој UI у одређени DOM елемент. За *React* апликације, ово типично укључује позив *ReactDOM.createRoot().render()*. *Mount* функција прима *props* објекат који може садржавати DOM елемент за *mounting* и додатне податке од основне конфигурације.

Unmount функција се позива када микрофронтенд више није активан (нпр. када корисник навигира на другу руту). Ова функција је критична за правилно чишћење ресурса: уклањање DOM елемената, отказивање *network* захтјева, брисање тајмера, и уклањање

event listener-а. Непотпуно чишћење може довести до *memory leak*-ова и неочекиваног понашања, посебно у дуготрајним сесијама гдје се микрофронтенди често монтирају и демонтирају.

Помоћне библиотеке радног оквира, попут *single-spa-react*, *single-spa-vue* и *single-spa-angular*, аутоматизују имплементацију функција животног циклуса за специфичне радне оквире. Ове библиотеке преузимају одговорност за правилну интеграцију са специфичним API-јима за *mounting* и *unmounting* радног оквира.

3.3.3. Рутирање и активација

Single-SPA користи концепт активирања функција за одређивање који микрофронтенди треба да буду активни у датом тренутку. Функција активирања је функција која прима тренутну локацију (URL) и враћа *boolean* вриједност која указује да ли микрофронтенд треба бити активан.

Регистрација микрофронтенда врши се путем *registerApplication* функције:

registerApplication({ name: 'app-name', app: () => import('url-to-app'), activeWhen: '/some-route' }).

Параметар *activeWhen* може бити текстуални тип података (провјерава да ли путања почиње датим текстом), низ текстова (OR логика), или функција за комплексну логику.

За декларативну дефиницију распореда, *single-spa-layout* библиотека омогућава коришћење HTML шаблона. Умјесто регистрације апликација у *JavaScript*-у, програмери дефинишу распоред у HTML-у коришћењем прилагођених елемената: <*single-spa-router*>, <*route*>, и <*application*>. Ово побољшава читљивост и омогућава нетехничким члановима тима да разумију структуру апликације.

Двонивовско рутирање је образац гдје *app shell* обавља само "грубо" рутирање на нивоу тима (нпр. */products/** припада тиму 1), док сваки микрофронтенд интерно обавља "фино" рутирање за своје странице. Ово смањује одговорност *app shell-а* и омогућава тимовима потпуну контролу над њиховим рутама без координације са другим тимовима.

3.3.4. Import Maps и дијељење зависности

Import Maps је веб стандард (подржан у свим модерним прегледачима) који омогућава контролу над резолуцијом *JavaScript* модула. Путем JSON конфигурације у <*script type="importmap"*> елементу, програмери мапирају голе(bare) module *specifier*-е (као "*react*") на стварне URL-ове.

У *Single-SPA* контексту, *Import Maps* се користе за двије сврхе. Прво, за дијељење заједничких библиотека: умјесто да сваки микрофронтенд *bundle*-ује *React*, сви референцирају исту CDN верзију дефинисану у *Import Map*-и. Друго, за конфигурацију локација микрофронтенда: логичка имена као "*@org/header*" мапирају се на URL-ове сервера који хостује тај микрофронтенд.

SystemJS је библиотека за учитавање која пружа полифил за *Import Maps* у старијим прегледачима који не подржавају нативне ES модуле. *Single-SPA* често користи *SystemJS* format за компатибилност, иако модерни прегледачи могу користити нативне ES модуле директно.

3.3.5. Комуникација између микрофронтенда

Пошто *Single-SPA* микрофронтенди дијеле исти *JavaScript* вријеме извршавања, доступно је више механизама за комуникацију. Најједноставнији приступ је коришћење URL-а као извора истине: стање се кодира у URL параметрима или путањи, а сваки микрофронтенд

реагује на промјене URL-а. Овај приступ има предност природне подршке за дубоко линковање и историју веб читача.

Prilagođeni događaji *(custom events)* су стандардни DOM механизам који омогућава *publish/subscribe* комуникацију. Микрофронтенд може емитовати догађај путем *window.dispatchEvent(new CustomEvent('eventName', { detail: data }))*, а други микрофронтенди могу ослушкивати путем *window.addEventListener('eventName', handler)*.

Реактивни приступ путем RxJS библиотеке омогућава напреднију комуникацију. *Event bus* микрофронтенд може излагати *BehaviorSubject* инстанце које представљају глобално стање. *BehaviorSubject* чува тренутну вриједност и емитује је новим претплатницима, што је идеално за управљање стањима. Остали микрофронтенди се претплаћују на ове *Observable*-е и аутоматски се ажурирају када се стање промијени.

3.3.6. Предности *Single-SPA* приступа

Агностицизам радног оквира је кључна предност *Single-SPA*. Исти оркестратор може управљати *React*, *Vue*, *Angular*, *Svelte* апликацијама, па чак и plain *JavaScript* модулима. Ово је посебно вриједно у сценаријима миграције гдје нови микрофронтенди могу користити модерни радни оквир док старе компоненте остају на старој технологији.

Унификована навигација омогућава корисницима добро искуство. Сви микрофронтенди дијеле исту историју у претраживачу, тако да дугмад за навигацију напред и назад и дубоко линковање функционишу очекивано. *Single-SPA* аутоматски управља интерцепцијом навигацијских догађаја и координацијом *mount/unmount* циклуса.

Зрелост екосистема значи да постоји обимна документација, подршка заједнице, и богат избор помоћних библиотека. *Single-spa-layout* за декларативно рутирање, *single-spa-leaked-globals* за детекцију изливања меморије, и бројни специфични адаптери за радни оквир доступни су као *npm* пакети.

Гранулирана контрола над животним циклусом омогућава имплементацију напредних образаца као што су *lazy* учитавање (учитавање кода тек када је потребан), пред учитавање, и паралелно *mounting* више микрофронтенда.

3.3.7. Недостаци *Single-SPA* приступа

Почетна сложеност конфигурације може бити застрашујућа за тимове нове у микрофронтенд архитектури. *Root-config*, *Import Maps*, функције животног цикуса, и адаптери радног оквира захтијевају разумијевање концепата који нису присутни у традиционалном развоју клијентских апликација.

Дијељени времена извршавања значи да грешка или изливање меморије у једном микрофронтенду може утицати на цијелу апликацију. За разлику од *iframe* изолације, лоше написан код једног тима може деградирати перформансе или стабилност апликација других тимова. Ово захтијева дисциплинован приступ квалитету кода и темељно тестирање.

CSS изолација није аутоматска. Пошто сви микрофронтенди дијеле исти DOM, глобални CSS селектори једног микрофронтенда могу утицати на елементе другог. Рјешења укључују *CSS Modules*, CSS-in-JS библиотеке, или строге naming конвенције. *Build* конфигурација захтијева пажњу. Микрофронтенди морају бити изграђени у формату компатибилном са *Single-SPA* (*SystemJS* или *ES modules*), са екстернализованим дијељеним зависностима. Ово може захтијевати прилагођавање постојећих *build* система.

3.4 Компаративна анализа приступа

Сљедећа табела сумира кључне карактеристике сва три приступа, омогућавајући брзо упоређивање на основу релевантних критеријума за одабир архитектуре.

Критеријум	***iframe***	***Module Federation***	***Single-SPA***
Ниво изолације	Потпуна	Дијељена	Дијељена
Слобода радног оквира	Потпуна	Ограничена	Потпуна
Дијељење зависности	Немогуће	Аутоматско	Путем *Import Maps*
Комуникација	postMessage	*Context/Props*	*Events/RxJS*
Сложеност имплементације	Ниска	Средња	Висока
Перформансе	Лошије	Боље	Боље
SEO подршка	Слаба	Добра	Добра
Погодност за старе системе	Одлична	Средња	Добра

Табела 3.1: Упоредни преглед приступа микрофронтенд интеграцији

Избор приступа треба бити вођен специфичним захтјевима пројекта. *Iframe* приступ је најпогоднији када је изолација примарни захтјев, посебно при интеграцији застарјелих система или компоненти трећих страна. *Module Federation* је оптималан избор за тимове који користе исти технолошки стек и желе максимизовати дијељење кода уз задржавање независних циклуса испоруке. *Single-SPA* је најфлексибилније рјешење за хетерогене тимове који требају агностицизам радног оквира и софистицирано управљање животним циклусом апликација.

4. Технологије коришћене у имплементацији

У овом поглављу представљене су технологије које су коришћене за имплементацију три различита приступа микрофронтенд архитектури. Свака имплементација користи специфичан скуп алата и библиотека који најбоље одговарају одабраном обрасцу интеграције. Заједничка основа свих пројеката укључује модерне *JavaScript* технологије, док се специфичности разликују у зависности од механизма композиције и комуникације.

4.1. Заједничке технологије

Сва три пројекта дијеле одређени скуп технологија који чини основу модерног развоја клијентских апликација. Ове технологије су одабране због своје поузданости, перформанси и широке подршке заједнице.

4.1.1. *Vite*

Vite је модеран алат за изградњу(*build*) клијентских апликација који је креирао Еван Ју, аутор *Vue.js* радног оквира. Назив долази од француске ријечи за "брзо", што одражава његову примарну карактеристику. За разлику од традиционалних алата попут *Webpack*-а који захтијевају *bundling* током развоја, *Vite* користи нативне ES модуле прегледача што резултира значајно бржим временом покретања развојног сервера.

Кључне предности *Vite* алата укључују тренутно покретање сервера без обзира на величину пројекта, *Hot Module Replacement* (HMR) који ажурира промјене у милисекундама, оптимизовану производну изградњу засновану на *Rollup*-у, те подршку за *TypeScript*, JSX и CSS модуле без додатне конфигурације. У контексту микрофронтенда, *Vite* омогућава сваком тиму независан развојни процес са минималним временом чекања.

4.1.2. *TypeScript*

TypeScript је надскуп *JavaScript*-а који додаје статичку типизацију. Развијен од стране *Microsoft*-а, *TypeScript* омогућава откривање грешака током компилације, што значајно побољшава поузданост кода. У микрофронтенд архитектури, *TypeScript* игра посебно важну улогу у дефинисању уговора између различитих модула и тимова.

Типови података који се дијеле између микрофронтенда, попут структуре порука у iframe приступу или интерфејса контекста у *Module Federation* приступу, дефинисани су путем *TypeScript* интерфејса. Ово осигурава да све стране поштују договорену структуру података и омогућава IDE подршку за аутоматско допуњавање и провјеру типова.

4.1.3. *TailwindCSS*

TailwindCSS је радни оквир заснован на помоћним CSS класама који омогућава брзу изградњу корисничких интерфејса без напуштања HTML-а. Умјесто писања прилагођеног CSS-а, програмери користе унапријед дефинисане класе директно у *markup*-у. Примјер класе "*flex items-center justify-between p-4*" комбинује *flexbox* распоред, центрирање елемената, распоређивање са размаком и *padding* у једној линији.

У микрофронтенд контексту, *TailwindCSS* пружа конзистентан дизајн систем који сви тимови могу користити. Како се класе примјењују директно на елементе, ризик од конфликта стилова између микрофронтенда је минималан. Поред тога, продукциони *build TailwindCSS*-а аутоматски уклања некоришћене класе, што резултира минималном величином CSS *bundle*-а.

4.1.4. *Google Books* API

Google Books API је REST сервис који омогућава приступ библиотеци књига компаније *Google*. У свим имплементацијама овај API служи као извор података о књигама,

укључујући наслове, ауторе, описе, слике насловница и информације о цијенама. API је одабран због своје бесплатне доступности, обимне базе података и једноставног интерфејса.

Типичан захтјев за претрагу књига има облик *GET https://www.googleapis.com/books/v1/volumes?q=javascript*, док се детаљи појединачне књиге добијају путем GET *https://www.googleapis.com/books/v1/volumes/{volumeId}*. Одговор укључује метаподатке као што су *volumeInfo* објекат са насловом, ауторима и описом, те *saleInfo* објекат са информацијама о цијени.

4.2. Технологије *iframe* приступа

Iframe приступ демонстрира технолошку разноликост као једну од кључних предности микрофронтенд архитектуре. Свака апликација у овом пројекту користи различит *JavaScript* радни оквир, а све комуницирају путем стандардизованог *postMessage* API-ја.

Компонента	Радни Оквир	Верзија	Улога
Container	*React*	18.3	Главна апликација
Book List	*Svelte*	5.15	Листа књига
Single Book	*Vue*	3.5	Детаљи књиге

Табела 4.1: *Технологије iframe приступа*

4.2.1. *React*

React је *JavaScript* библиотека за изградњу корисничких интерфејса коју је развио *Facebook* (сада *Meta*). У *iframe* имплементацији, *React* се користи за главну (*container)* апликацију која служи као оркестратор цијелог система. Главна (*Container)* апликација је одговорна за рендеровање *iframe* елемената, управљање глобалним стањем као што је корпа за куповину, те рутирање порука између микрофронтенда.

React 18, верзија коришћена у пројекту, доноси побољшања у конкурентном рендеровању и аутоматском батхирању ажурирања стања. *Hooks* API, уведен у *React 16.8*, користи се интензивно за управљање стањем (*useState*), ефектима (*useEffect*) и меморизацијом (*useMemo*, *useCallback*).

4.2.2. *Svelte*

Svelte је компајлер који претвара декларативне компоненте у ефикасан *JavaScript* код. За разлику од *React*-а и *Vue*-а који користе виртуелни DOM, *Svelte* компајлира компоненте у императивни код који директно манипулише DOM-ом. Ово резултира мањим *bundle*-овима и бржим извршавањем.

У имплементацији, *Svelte* се користи за *Book List* микрофронтенд. *Svelte 5*, најновија верзија, уводи "*runes*" систем за реактивност који замјењује претходну магичну *$:* синтаксу. Функције животног циклуса *onMount* и *afterUpdate* користе се за иницијализацију слушалаца порука и аутоматско прилагођавање висине *iframe*-а.

4.2.3. *Vue.js*

Vue.js је прогресивни *JavaScript* радни оквир за изградњу корисничких интерфејса. У имплементацији се користи за *Single Book* микрофронтенд који приказује детаље појединачне књиге. *Vue 3*, верзија коришћена у пројекту, доноси *Composition* API као алтернативу *Options* API-ју, побољшану *TypeScript* подршку и оптимизован виртуелни DOM.

Single Book компонента демонстрира *Vue* реактивност кроз *data()* функцију за дефинисање стања, методе за обраду корисничких интеракција, те хукове животног циклуса *mounted* и *updated* за интеграцију са системом за размјену порука.

4.2.4. Pnpm радни простор(монорепозиторијум)

pnpm је алтернативни менаџер пакета за *Node.js* који се истиче ефикасним коришћењем дисковног простора путем складишта за адресабилни садржај. *pnpm* радни простор омогућава организацију више повезаних пакета у једном репозиторијуму, што је познато као монорепозиторијумска структура.

У *iframe* пројекту, монорепозиторијумска структура омогућава дијељење заједничког кода (*shared* фолдер) између три апликације без потребе за објављивањем *npm* пакета. Конфигурација *pnpm-workspace.yaml* дефинише да су сви пројекти у *apps/** директоријуму дио радног простора. Ово омогућава референцирање дијељеног(*shared)* модула путем релативних путања.

4.2.5. *postMessage* API

window.postMessage() је веб API који омогућава сигурну комуникацију између прозора различитог поријекла. У *iframe* имплементацији, овај API је централни механизам за размјену података између главне апликације и угнијеждених микрофронтенда.

Имплементација укључује неколико безбједносних мјера. Валидација поријекла осигурава да се поруке примају само од дозвољених домена дефинисаних путем варијабли окружења(*environment*). Валидација структуре поруке провјерава да примљени подаци одговарају очекиваном формату. Забрана *wildcard*-а спречава слање порука свим прозорима.

Типови порука укључују *SHOW_SINGLE_BOOK* за приказ детаља књиге, *ADD_TO_CART* за додавање у корпу, *SEARCH_BOOK_LIST* за претрагу, и *RESIZE_IFRAME* за аутоматско прилагођавање висине *iframe*-а.

4.3. Технологије *Module Federation* приступа

Module Federation приступ фокусира се на дијељење кода и зависности у времену извршавања. Све апликације користе исти технолошки стек (*React*), што омогућава дубљу интеграцију и дијељење *React* контекста за управљање стањем.

Компонента	Тип	Излаже	Конзумира
host-react	*Host*	*GlobalContext*	*bookList, singleBook, shared*
book-list-react	*Remote*	*App*	*host, shared*
single-book-react	*Remote*	*App*	*host, shared*
shared-react	*Remote*	*Button, Search, ErrorBoundary*	-

Табела 4.2: Технологије *Module Federation* приступа

4.3.1. *Webpack Module Federation*

Module Federation је функционалност *Webpack 5 bundler*-а која омогућава динамичко дијељење кода између независно изграђених апликација током времена извршавања. Уведен 2020. године, овај механизам представља значајан напредак у односу на претходне методе дијељења кода које су захтијевале интеграцију у времену изградње (*buildtime*).

Кључни концепти *Module Federation* укључују и апликацију која учитава удаљене (*remote)* модуле, *Remote* апликације које излажу своје модуле путем *remoteEntry.js* датотеке, *Exposes* конфигурацију која дефинише које модуле апликација чини доступним другима, те *shared* конфигурацију која омогућава дијељење зависности попут *React*-а како би се избјегло дуплирање.

4.3.2. *@originjs/vite-plugin-federation*

Пошто је оригинални *Module Federation* дио *Webpack*-а, за *Vite* базиране пројекте користи се *@originjs/vite-plugin-federation*. Овај додатак омогућава исту функционалност *Module Federation*-а у *Vite* екосистему, с тим да генерише компатибилне *remoteEntry.js* датотеке.

Конфигурација додатака (*plugin*) укључује параметар имена (*name*) који идентификује апликацију, име фајла (*filename)* који дефинише име излазне датотеке (типично *remoteEntry.js*), *exposes* објекат који мапира путање модула на локалне датотеке, *remotes* објекат који дефинише URL-ове удаљених модула, и *shared* низ који листа зависности за дијељење.

4.3.3. *React Context* API

React Context API је механизам за прослеђивање података кроз стабло компоненти без потребе за ручним прослеђивањем пропова на сваком нивоу. У *Module Federation* имплементацији, контекст се користи као централно мјесто за глобално стање апликације.

GlobalContext, дефинисан у *host* апликацији и изложен путем *Module Federation*, садржи стање корпе (*cart*), тренутно одабрану књигу (*selectedBook*), текст претраге (*searchText*), те функције за модификацију овог стања. *Remote* апликације увозе *useGlobalContext hook* и тако добијају приступ глобалном стању без директне комуникације са *host*-ом.

4.3.4. *React Suspense* и *Lazy* учитавање

React.lazy() и *Suspense* компонента омогућавају динамичко учитавање компоненти. У *Module Federation* контексту, ово је посебно важно јер се *remote* модули учитавају преко мреже. *Suspense* компонента приказује *fallback* садржај док се асинхрона компонента учитава.

У имплементацији, *BookList* и *SingleBook* компоненте се учитавају путем *React.lazy()* који увози *remote* модуле. Свака *lazy* учитана компонента је омотана у *Suspense* који приказује индикатор учитавања током учитавања. *ErrorBoundary* компонента хвата грешке које могу настати ако *remote* модул није доступан.

4.4. Технологије *Single-SPA* приступа

Single-SPA приступ користи наменски радни оквир за оркестрацију микрофронтенда са фокусом на управљање животним циклусом и централизовано рутирање. Комуникација између апликација остварује се путем реактивног *event bus*-а заснованог на RxJS библиотеци.

Репозиторијум	Порт	Улога
root-config	3005	Оркестратор, HTML шаблон, рутирање
event-bus	5173	RxJS *event bus* за комуникацију
header	3008	Навигација, претрага, корпа
book-list	3006	Листа књига
single-book	3007	Детаљи књиге
footer	3009	Подножје странице

Табела 4.3: Структура *Single-SPA* пројекта

4.4.1. *Single-SPA* радни оквир

Single-spa је *JavaScript* радни оквир специјално дизајниран за изградњу микрофронтенд апликација. Његова примарна функција је оркестрација више независних клијентских апликација унутар јединственог контејнера. Радни оквир управља животним циклусом сваке апликације, укључујући учитавање, монтирање и демонтирање на основу тренутног URL-а.

Кључна карактеристика *single-spa* је агностицизам радног оквира. Исти оркестратор може управљати апликацијама писаним у *React*-у, *Vue*-у, *Angular*-у, *Svelte*-у или чистом *JavaScript*-у. Ово се постиже стандардизованим интерфејсом који свака апликација мора имплементирати: *bootstrap*, *mount* и *unmount* функције животног циклуса.

4.4.2. *single-spa-layout*

single-spa-layout је проширење које омогућава декларативну дефиницију распореда микрофронтенда путем HTML шаблона. Умјесто императивне регистрације апликација у *JavaScript*-у, програмери дефинишу распоред у посебном HTML документу.

Layout HTML користи прилагођене елементе попут <*single-spa-router*> за дефинисање контејнера рутирања, <*route*> за дефинисање рута са атрибутом *path* или *default*, и <*application*> за позиционирање микрофронтенда са атрибутом *name*. Овај приступ олакшава разумијевање структуре апликације и омогућава нетехничким члановима тима да схвате распоред.

4.4.3. *Import Maps*

Import Maps је веб стандард који омогућава контролу над резолуцијом *JavaScript* модула у прегледачу. Путем JSON конфигурације унутар <*script type="importmap"*> тага, програмери дефинишу мапирање између *bare module specifier*-а (нпр. "*react*") и стварних URL-ова.

У *Single-SPA* имплементацији, *Import Maps* се користе за двије сврхе. Прво, за дијељење библиотека попут *React*-а и *single-spa* између свих микрофронтенда без дуплирања кода. Друго, за мапирање логичких имена микрофронтенда (нпр. "*@single-spa/book-list*") на њихове стварне URL-ове. Ово омогућава промјену локације микрофронтенда без модификације кода који их референцира.

4.4.4. *single-spa-react*

single-spa-react је adapter библиотека која олакшава интеграцију *React* апликација са *single-spa* оркестратором. Она аутоматски генерише потребне функције животног циклуса на основу *React* компоненте.

singleSpaReact функција прима конфигурациони објекат који укључује *React* и *ReactDOMClient* референце, *rootComponent* као главну *React* компоненту, и опционо *errorBoundary* за хватање грешака. Функција враћа објекат са *bootstrap*, *mount* и *unmount* функцијама животног циклуса које *single-spa* очекује.

4.4.5. *RxJS (Reactive Extensions for JavaScript)*

RxJS је библиотека за реактивно програмирање која имплементира *Observer* образац. У *Single-SPA* имплементацији, *RxJS* се користи за имплементацију *event bus*-а који омогућава комуникацију између микрофронтенда без директних зависности.

BehaviorSubject, тип *Observable*-а који чува тренутну вриједност и емитује је новим претплатницима, користи се за три стања: *cartState$* за праћење садржаја корпе, *searchState$* за текст претраге, и *selectedBookState$* за тренутно одабрану књигу. Микрофронтенди се претплаћују на ове *Observable*-е и аутоматски реагују на промјене стања.

Event bus модул се монтира као први микрофронтенд и излаже своје функције на глобалном *window* објекту (*window.eventBus*). Остали микрофронтенди приступају овим функцијама за слање акција и претплату на промјене стања.

4.5. Упоредни преглед технологија

Одабир технологија у сваком приступу директно утиче на карактеристике резултујуће архитектуре. Сљедећа табела сумира кључне разлике у технолошким одлукама.

Аспект	iframe	Module Federation	Single-SPA
Алалт за изградњу	*Vite*	*Vite* + *MF* додатак	*Vite*
Радни оквир	*React, Svelte, Vue*	*React* (сви)	*React* (сви)
Комуникација	*postMessage API*	*React Context*	*RxJS Event Bus*
Структура репоа	монорепозиторијум *(pnpm)*	Засебни репои	Засебни репои
Дијељење кода	*shared* фолдер	Вријеме извршавања (MF)	*Import Maps*
Изолација	Потпуна (*iframe*)	Дијељена	Дијељена

Табела 4.4: Упоредни преглед технологија

Избор технологија у великој мјери одређује предности и ограничења сваког приступа. *Iframe* приступ користи разноликост радних оквира за демонстрацију технолошке агностичности, али захтијева експлицитну комуникацију путем *postMessage*-а. *Module Federation* омогућава дубоку интеграцију и дијељење зависности, али ограничава избор радних оквира. *Single-SPA* нуди баланс између флексибилности и структуре уз помоћ реактивне комуникације.

5. Концепт Ријешења

У овом поглављу представљен је концепт апликације која служи као основа за практичну демонстрацију три различита приступа микрофронтенд архитектури. Апликација "*B-World*" је е-продавница за продају књига која омогућава корисницима претрагу, преглед детаља и куповину књига. Идентичне функционалности имплементиране су кроз *iframe*, *Module Federation* и *Single-SPA* приступе, што омогућава објективну упоредну анализу карактеристика сваког приступа.

5.1. Опис апликације

B-World је веб апликација за електронску трговину књигама дизајнирана као демонстрациона платформа за испитивање микрофронтенд архитектуре. Назив "*B-World*" представља скраћеницу од "*Book World*", одражавајући примарну функцију апликације као виртуелне књижаре. Апликација је намјерно дизајнирана са умјереном сложеношћу која је довољна за демонстрацију кључних аспеката микрофронтенд архитектуре, укључујући композицију корисничког интерфејса, комуникацију између модула и управљање дијељеним стањем.

Концептуално, апликација симулира реални е-продавница сценарио гдје купац може претраживати каталог производа, прегледати детаљне информације о појединачним производима, и додавати жељене производе у корпу за куповину. Иако апликација не укључује комплетан процес продаје са плаћањем, функционалности које имплементира су довољне за демонстрацију свих релевантних образаца микрофронтенд архитектуре.

5.1.1 Циљеви апликације

Примарни циљ *B-World* апликације је пружање платформе за упоредну анализу три различита приступа микрофронтенд архитектури. Ова анализа укључује евалуацију сложености имплементације сваког приступа, мјерење перформанси у погледу времена учитавања и одзива корисничког интерфејса, процјену погодности за независан развој од стране вишеструких тимова, и идентификацију предности и недостатака сваког приступа у контексту реалних захтјева.

Секундарни циљ је демонстрација практичне примјене теоријских концепата представљених у претходним поглављима. Кроз имплементацију идентичних функционалности у три различита приступа, омогућена је директна визуелизација разлика у архитектонским одлукама и њиховим посљедицама.

5.1.2. Кориснички профил

Циљна група корисника *B-World* апликације су љубитељи књига који желе пронаћи и купити књиге путем интернета. Типичан корисник је особа која зна шта тражи (претрага по кључним ријечима) или жели истражити понуду (прегледање листе књига). Корисник очекује брз одзив интерфејса, јасан приказ информација о књизи, и једноставан процес додавања жељених књига у корпу.

Иако апликација нема имплементирану аутентификацију корисника, дизајнирана је са претпоставком да би у продукцијском окружењу постојали регистровани корисници са персонализованим искуством, историјом куповине и препорукама заснованим на претходним активностима.

5.2. Функционални захтјеви

Функционални захтјеви дефинишу специфичне могућности које апликација мора пружити корисницима. Ови захтјеви су идентични за све три имплементације, осигуравајући поуздану основу за упоредну анализу.

ID	Захтјев	Опис
ФЗ-01	Претрага књига	Корисник може унијети текст претраге и добити листу књига које одговарају упиту
ФЗ-02	Приказ листе књига	Систем приказује листу књига са насловом, аутором и сликом насловнице
ФЗ-03	Приказ детаља књиге	Корисник може видјети детаљне информације о одабраној књизи укључујући опис, издавача и цијену
ФЗ-04	Додавање у корпу	Корисник може додати књигу у корпу за куповину са одабраном количином
ФЗ-05	Преглед корпе	Корисник може видјети садржај корпе са укупном цијеном
ФЗ-06	Брисање из корпе	Корисник може уклонити ставке из корпе
ФЗ-07	Навигација	Корисник може навигирати између листе књига и детаља одабране књиге

Табела 5.1: Функционални захтјеви апликације

5.3. Архитектура система

Архитектура *B-World* апликације слиједи принципе микрофронтенд архитектуре, дијелећи кориснички интерфејс на логичке цјелине које се могу независно развијати и постављати. Без обзира на одабрани приступ интеграције, концептуална подјела на микрофронтенде остаје конзистентна.

5.3.1. Подјела на микрофронтенде

Апликација је подијељена на неколико логичких цјелина, од којих свака представља потенцијални домен одговорности засебног тима. Ова подјела одражава принцип вертикалног резања функционалности гдје сваки микрофронтенд посједује комплетан стек од корисничког интерфејса до интеграције са *backend* сервисима.

Микрофронтенд	Одговорност	Функционалности
Container/Host	Оркестрација и композиција	Распоред странице, рутирање, глобално стање
Header	Навигација и претрага	Лого, поље за претрагу, приказ корпе
Book List	Каталог књига	Листа књига, филтрирање, селекција
Single Book	Детаљи производа	Информације о књизи, додавање у корпу
Footer	Подножје странице	Информације о апликацији, линкови

Табела 5.2: Подјела на микрофронтенде

Container (или *Host* у терминологији *Module Federation*, односно *root-config* у *Single-SPA*) представља централну тачку апликације која координира остале микрофронтенде. Овај модул је одговоран за иницијално учитавање апликације, дефинисање распореда елемената на страници, и управљање глобалним стањем као што је садржај корпе за куповину.

Header микрофронтенд обезбјеђује конзистентну навигацију кроз апликацију. Укључује лого апликације, поље за унос текста претраге, и визуелни индикатор тренутног броја ставки у корпи. Овај модул комуницира са осталим микрофронтендима како би пренио упите за претрагу и примио ажурирања о стању корпе.

Book List микрофронтенд је одговоран за приказ каталога књига. Прима параметре претраге од *Header* модула, извршава API позиве према *Google Books* сервису, и рендерује резултате у облику картица са основним информацијама о свакој књизи. Клик на појединачну књигу активира *Single Book* микрофронтенд.

Single Book микрофронтенд приказује детаљне информације о одабраној књизи. Укључује насловницу веће резолуције, комплетан опис, информације о аутору и издавачу, те дугме за додавање у корпу. Овај модул комуницира са *Container*-ом за ажурирање стања корпе.

5.3.2. Ток података

Ток података у апликацији слиједи једносмјерни образац (*unidirectional data flow*) гдје стање тече од родитељских компоненти према дјеци, а догађаји се пропагирају у супротном смјеру. Овај образац, популаризован од стране *React* екосистема, олакшава праћење промјена стања и дебаговање.

Примарни токови података у апликацији укључују ток претраге који почиње када корисник унесе текст у поље за претрагу у *Header* модулу, текст се прослеђује *Book List* модулу, *Book List* извршава API позив и ажурира приказану листу књига. Ток селекције књиге почиње када корисник кликне на књигу у листи, идентификатор књиге се прослеђује *Single Book* модулу, *Single Book* преузима детаље и приказује их. Ток корпе се активира када корисник кликне дугме за додавање у корпу, подаци о књизи се прослеђују *Container* модулу који ажурира глобално стање корпе, *Header* модул прима ажурирање и освјежава број ставки.

5.3.3. Варијације архитектуре по приступу

Иако концептуална архитектура остаје иста, сваки приступ имплементира интеграцију на различит начин. У *iframe* приступу, *Container* апликација рендерује *iframe* елементе за *Book List* и *Single Book* модуле. Комуникација се одвија путем *postMessage* API-ја са структурираним порукама које укључују тип акције и *payload* податке.

У *Module Federation* приступу, *Host* апликација динамички учитава *Remote* модуле током времена извршавања. Комуникација се одвија путем дијељеног *React Context*-а који *Host* излаже, а *Remote* модули конзумирају. Ово омогућава директан приступ глобалном стању без експлицитне размјене порука.

У *Single-SPA* приступу, *root-config* региструје све микрофронтенде и управља њиховим животним циклусом. Засебан *event-bus* микрофронтенд обезбјеђује RxJS *Observable*-е за реактивну комуникацију. Микрофронтенди се претплаћују на релевантне *Observable*-е и аутоматски реагују на промјене стања.

5.4. Извор података - *Google Books API*

Google Books API служи као примарни извор података о књигама у апликацији. Овај REST API омогућава приступ обимној бази података књига која укључује метаподатке, описе, слике насловница и информације о цијенама за милионе наслова.

5.4.1. Структура API-ја

Google Books API користи *RESTful* архитектуру са JSON форматом за размјену података. Основна рута за претрагу је *https://www.googleapis.com/books/v1/volumes* који прихвата параметар претраге *q* за текст претраге. Одговор укључује листу волумена (књига) са пагинацијом.

Рута за преузимање детаља појединачне књиге има формат *https://www.googleapis.com/books/v1/volumes/{volumeId}* гдје *volumeId* представља јединствени идентификатор књиге добијен из резултата претраге. Ова рута враћа комплетне информације о књизи укључујући проширени опис и све доступне метаподатке.

Поље	Тип	Опис
id	*string*	Јединствени идентификатор књиге
volumeInfo.title	*string*	Наслов књиге
volumeInfo.authors	*string[]*	Листа аутора
volumeInfo.description	*string*	Опис књиге (може садржавати HTML)
volumeInfo.imageLinks	*object*	URL-ови слика насловнице (минијатурна слика, мала, средња)
volumeInfo.publisher	*string*	Назив издавача
volumeInfo.publishedDate	*string*	Датум издавања
saleInfo.listPrice	*object*	Цијена књиге (*amount*, *currencyCode*)

Табела 5.3: Структура одговора *Google Books API-ja*

5.4.2. Обрада података

Подаци добијени од *Google Books* API-ја захтијевају одређену обраду прије приказивања корисницима. Опис књиге (*volumeInfo.description*) може садржавати HTML тагове које је потребно санитизовати како би се спријечили XSS (*Cross-Site Scripting*) напади. У имплементацији се користи функција *stripHTML* која уклања све HTML тагове и враћа чист текст.

Слике насловница долазе у различитим резолуцијама. За листу књига користи се минијатурна слика(*thumbnail*), док се за приказ детаља користи средња или велика верзија ако је доступна. Апликација такође обрађује случајеве када слика није доступна, приказујући замјенску слику.

Информације о цијени нису увијек доступне за све књиге. Апликација провјерава постојање *saleInfo.listPrice* објекта и приказује одговарајућу поруку када цијена није доступна. За књиге са доступном цијеном, приказује се износ у одговарајућој валути.

5.5. Кориснички сценарији

Кориснички сценарији (*use cases*) описују типичне интеракције корисника са апликацијом. Ови сценарији су коришћени као основа за дизајн корисничког интерфејса и валидацију имплементиране функционалности.

5.5.1. Сценарио: Претрага и куповина књиге

Корисник отвара апликацију са намјером да пронађе и купи књигу о програмирању у *JavaScript*-у. Прво, корисник уноси "*JavaScript*" у поље за претрагу у заглављу странице и притиска *Enter* или кликће на дугме за претрагу. Систем приказује листу књига које одговарају упиту, са насловом, аутором и насловницом за сваку књигу.

Корисник прегледа резултате и кликће на књигу која га занима. Систем приказује детаљне информације о одабраној књизи укључујући комплетан опис, информације о издавачу и цијену. Корисник одлучује да купи књигу и кликће на дугме "Додај у корпу". Систем потврђује додавање визуелном индикацијом и ажурира број ставки у корпи приказан у заглављу.

Корисник жели додати још једну књигу па се враћа на листу претраге. Понавља процес селекције и додавања. Коначно, корисник кликће на иконицу корпе да прегледа садржај прије финализације куповине.

5.5.2. Сценарио: Истраживање каталога

Корисник нема конкретну књигу на уму већ жели истражити понуду. Отвара апликацију и уноси општи појам попут "*fiction*" или "*science*". Прегледа приказану листу књига скролујући кроз резултате. Кликће на неколико књига редом да прочита описе и упореди опције.

Током истраживања, корисник може додати занимљиве књиге у корпу за каснију куповину или напустити детаљни приказ и наставити прегледање листе. Овај сценарио наглашава важност брзе навигације између листе и детаља, као и потребу да се стање (одабрана књига, садржај корпе) очува током навигације.

5.6. Дизајн корисничког интерфејса

Кориснички интерфејс *B-World* апликације дизајниран је са фокусом на једноставност, јасноћу и конзистентност. *TailwindCSS* радни оквир обезбјеђује униформан визуелни језик кроз све микрофронтенде, осигуравајући да корисник доживљава апликацију као кохерентну цјелину упркос техничкој подјели на независне модуле.

5.6.1. Распоред странице

Распоред странице слиједи уобичајени образац за е-продавница апликацију. Заглавље (*Header*) је фиксирано на врху екрана и садржи лого апликације позициониран лијево, поље за претрагу центрирано, и индикатор корпе позициониран десно. Овај распоред је конзистентан на свим страницама апликације.

Главни садржај заузима централни дио екрана испод заглавља. За листу књига, садржај је организован у мрежни распоред (*grid*) са прилагодљивим бројем колона у зависности од ширине екрана. За детаље књиге, садржај је организован у двоколонски распоред са сликом лијево и текстуалним информацијама десно.

Подножје (*Footer*) је позиционирано на дну странице и садржи информације о апликацији, линкове и ауторска права. Подножје је видљиво када корисник скролује до краја садржаја.

5.6.2. Компоненте интерфејса

Картица књиге (*Book Card*) је основна компонента за приказ књиге у листи. Садржи слику насловнице фиксних димензија, наслов књиге (ограничен на двије линије са *truncation*), име аутора, и опционо цијену. Картица има *hover* ефекат који визуелно указује на могућност клика.

Панел детаља књиге приказује проширене информације укључујући већу слику насловнице, комплетан наслов, листу аутора, издавача са годином издавања, комплетан опис (са *scroll*-ом ако је предугачак), цијену истакнуту већим фонтом, и дугме за додавање у корпу.

Компонента корпе приказује листу додатих књига са могућношћу уклањања појединачних ставки. Свака ставка приказује мини слику насловнице, наслов, количину и цијену. На дну је приказана укупна цијена свих ставки.

5.6.3. Респонзивни дизајн

Апликација је дизајнирана да буде респонзивна, прилагођавајући се различитим величинама екрана. На великим екранима (*desktop*), листа књига приказује четири до пет картица по реду. На средњим екранима (*tablet*), број колона се смањује на три. На малим екранима (*mobile*), картице се приказују у једној или двије колоне.

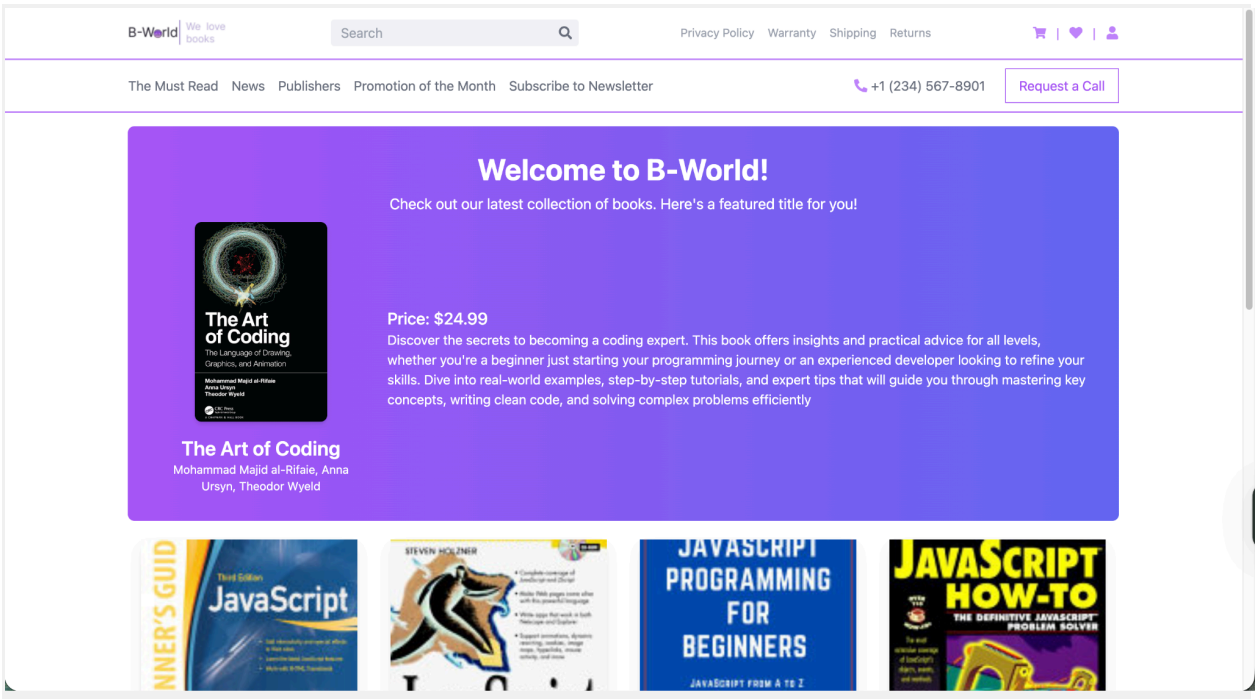
Детаљни приказ књиге такође се прилагођава. На великим уређајима, слика и информације су раме уз раме. На мобилним уређајима, слика се приказује изнад текстуалних информација у једноколонском распореду.

TailwindCSS utility класе као што су *sm:, md:, lg:* и *xl:* користе се за дефинисање стилова специфичних за одређене *breakpoint*-е. Ово омогућава декларативан приступ респонзивном дизајну директно у *markup*-у без потребе за засебним CSS датотекама.

6. Имплементација

У овом поглављу детаљно је представљена имплементација *B-World* апликације кроз три различита приступа микрофронтенд архитектури. За сваки приступ приказана је структура пројекта, кључни дијелови изворног кода, и објашњење архитектуралних одлука. Циљ је демонстрирати практичну примјену теоријских концепата представљених у претходним поглављима и омогућити читаоцу дубље разумијевање специфичности сваког приступа.

Сва три пројекта су јавно доступна на *GitHub* платформи, омогућавајући читаоцима да прегледају комплетан изворни код и самостално покрену апликације. Репозиторијуми су организовани према најбољим праксама за микрофронтенд архитектуру, укључујући јасну раздвојеност модула, документацију, и конфигурационе датотеке за развојна и продукциона окружења.



Слика 6.1: Почетни екран B-World апликације

6.1. Имплементација *iframe* приступа

Iframe имплементација демонстрира технолошку агностичност микрофронтенд архитектуре кроз коришћење три различита *JavaScript* радног оквира: *React* за контејнер апликацију, *Svelte* за листу књига, и *Vue.js* за приказ детаља књиге. Ова разноликост технологија наглашава једну од кључних предности *iframe* приступа - потпуну слободу у избору технолошког стека за сваки микрофронтенд.

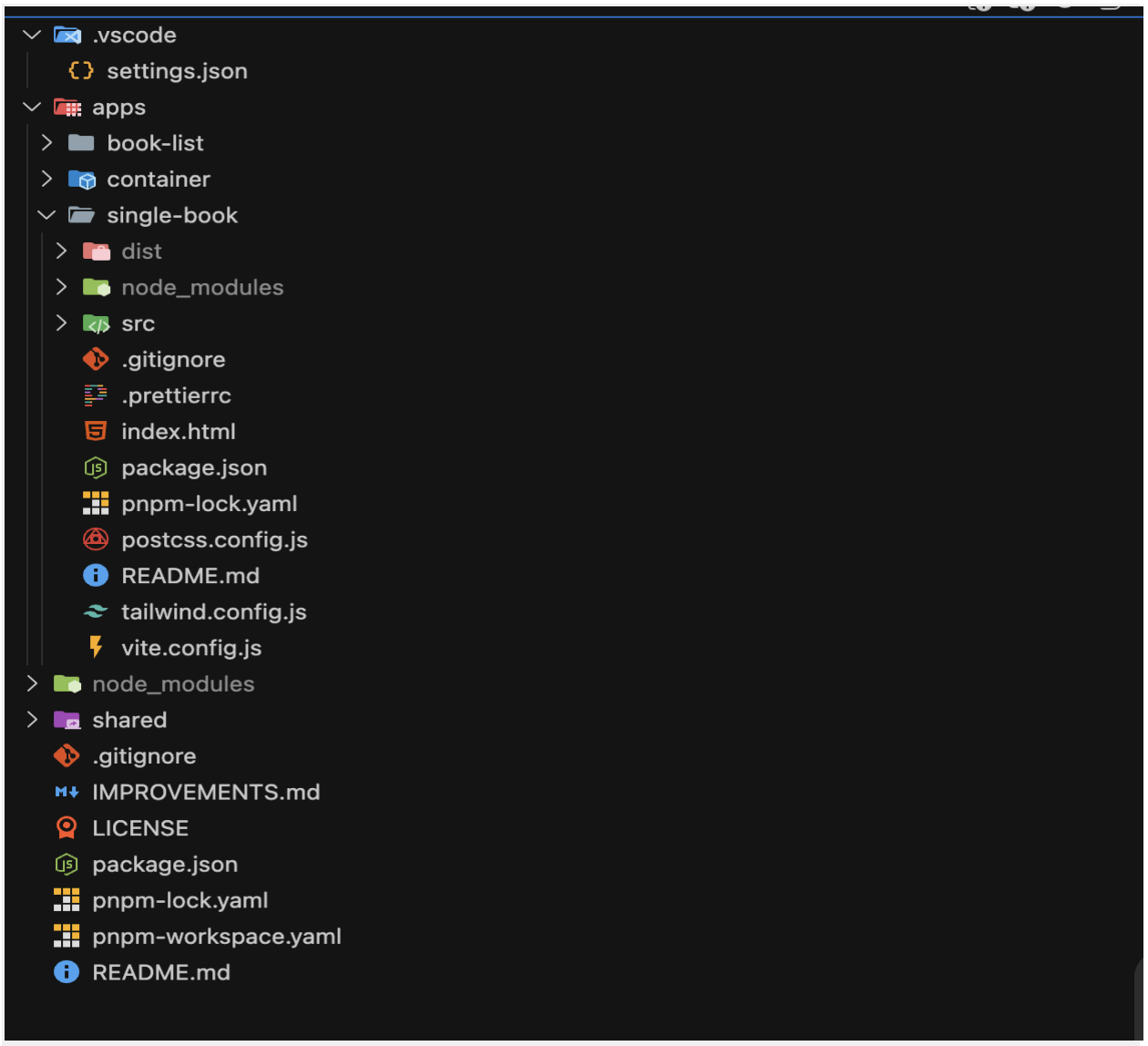
6.1.1. Структура пројекта

Пројекат је организован као монорепозиторијум коришћењем *pnpm* радног простора, што омогућава дијељење кода између апликација без потребе за објављивањем *npm* пакета. Структура директоријума јасно одваја појединачне апликације у *apps/* директоријуму, док *shared/* директоријум садржи заједнички код који користе све апликације.

```
iframe-micro-frontends/
├── apps/
│   ├── container/          # React 18.3 - главна апликација
│   │   ├── src/
│   │   │   ├── components/
│   │   │   │   ├── Header.tsx
│   │   │   │   ├── Footer.tsx
│   │   │   │   ├── Cart.tsx
│   │   │   │   └── IframeErrorBoundary.tsx
│   │   │   ├── App.tsx
│   │   │   └── main.tsx
│   │   ├── package.json
│   │   └── vite.config.ts
│   ├── book-list/          # Svelte 5.15 - листа књига
│   │   ├── src/
│   │   │   ├── components/
│   │   │   │   └── BookCard.svelte
│   │   │   ├── App.svelte
│   │   │   └── main.ts
│   │   └── package.json
│   └── single-book/        # Vue 3.5 - детаљи књиге
│       ├── src/
│       │   ├── components/
│       │   │   └── BookDetail.vue
│       │   ├── App.vue
│       │   └── main.ts
│       └── package.json
├── shared/                  # Дијељени TypeScript код
│   ├── communication.ts     # postMessage функције
│   ├── types.ts            # TypeScript интерфејси
│   ├── config.ts           # URL конфигурација
│   ├── sanitization.ts     # HTML санитизација
│   └── utils.ts            # Помоћне функције
├── pnpm-workspace.yaml
└── package.json
```

Листинг 6.1: Структура *iframe* пројекта

Конфигурација *pnpm* радног простора дефинисана је у *pnpm-workspace.yaml* датотеци која специфицира да су све апликације у *apps/* директоријуму и *shared/* директоријум дио истог радног простора. Ово омогућава референцирање *shared* модула путем релативних путања без потребе за публикацијом на *npm*.



Слика 6.2: Визуелни приказ монорепозиторијумске структуре *iframe* пројекта

6.1.2. Дијељени модул за комуникацију

Централни елемент *iframe* имплементације је *shared/communication.ts* модул који обезбјеђује апстракцију над *postMessage* API-јем. Овај модул дефинише функције за сигурно слање и примање порука између *iframe*-ова, укључујући валидацију поријекла порука као критичну безбједносну мјеру.

Функција *isOriginAllowed()* провјерава да ли је поријекло примљене поруке на листи дозвољених поријекла. Ова провјера спречава злонамјерне скрипте да шаљу лажне поруке апликацији. Листа дозвољених поријекла се конфигурише на основу окружења (развојно или продукционо).

```
// shared/communication.ts

import { getAllowedOrigins } from './config';

export interface MessagePayload {
  action: string;
  payload?: Record<string, unknown>;
}

// Провјера дозвољеног поријекла
export function isOriginAllowed(origin: string): boolean {
  const allowedOrigins = getAllowedOrigins();
  return allowedOrigins.includes(origin);
}

// Валидација структуре поруке
export function isValidMessage(message: unknown): message is MessageFormat {
  return (
    typeof message === 'object' &&
    message !== null &&
    'type' in message &&
    'data' in message
  );
}

// Слање поруке - забрањен wildcard из безбједносних разлога
export function sendMessage(
  targetWindow: Window,
  targetOrigin: string,
  messageType: string,
  payload: MessagePayload
): void {
  if (targetOrigin === '*') {
    throw new Error('Wildcard targetOrigin is not allowed');
  }
  targetWindow.postMessage({ type: messageType, data: payload }, targetOrigin);
}

// Регистрација слушаоца са cleanup функцијом
export function registerMessageListener(
  messageType: string,
  callback: (data: MessagePayload) => void
): () => void {
  function messageHandler(event: MessageEvent): void {
    if (!isOriginAllowed(event.origin)) return;
    if (!isValidMessage(event.data)) return;
    if (event.data.type === messageType) {
      callback(event.data.data);
    }
  }
  window.addEventListener('message', messageHandler);
  return () => window.removeEventListener('message', messageHandler);
}
```

Листинг 6.2: Валидација поријекла и слање порука

Типови порука су дефинисани као константе у *shared/types.ts* датотеци. *TypeScript*-ов *const assertion* (*'as const'*) омогућава да компајлер третира вриједности као литералне типове, обезбјеђујући *type-safety* при раду са порукама.

```
// shared/types.ts

export const MESSAGE_TYPE = 'MICROFRONTEND_MESSAGE';

export const MESSAGE_ACTIONS = {
  SHOW_SINGLE_BOOK: 'SHOW_SINGLE_BOOK',
```

```
  ADD_TO_CART: 'ADD_TO_CART',
  SEARCH_BOOK_LIST: 'SEARCH_BOOK_LIST',
  RESIZE_IFRAME: 'RESIZE_IFRAME',
} as const;

export const FRAME_NAMES = {
  BOOK_LIST: 'book-list-frame',
  SINGLE_BOOK: 'single-book-frame',
} as const;

export interface CartItem {
  bookId: string;
  title: string;
  authors?: string[];
  thumbnail?: string;
  price?: number;
  quantity: number;
}
```

Листинг 6.3: Дефиниција типова порука

6.1.3. *Container* апликација (*React*)

Container апликација је имплементирана у *React*-у и служи као централна тачка координације. Њена примарна одговорност је рендеровање *iframe* елемената, рутирање порука између микрофронтенда, управљање глобалним стањем (корпа), и пружање заједничких UI елемената (*Header*, *Footer*).

```
// apps/container/src/App.tsx

import { useEffect, useState, useCallback } from 'react';
import { registerMessageListener, sendMessage, getAppOrigin } from
'../../../shared/communication';
import { MESSAGE_TYPE, MESSAGE_ACTIONS, FRAME_NAMES, CartItem } from '../../../shared/types';
import Header from './components/Header';
import Footer from './components/Footer';
import IframeErrorBoundary from './components/IframeErrorBoundary';

// Мапирање акција на циљне iframe-ове
const IFRAME_ACTION_ROUTER = {
  [MESSAGE_ACTIONS.SHOW_SINGLE_BOOK]: {
    origin: getAppOrigin('single-book'),
    frameName: FRAME_NAMES.SINGLE_BOOK,
  },
  [MESSAGE_ACTIONS.SEARCH_BOOK_LIST]: {
    origin: getAppOrigin('book-list'),
    frameName: FRAME_NAMES.BOOK_LIST,
  },
};

export default function App() {
  const [cart, setCart] = useState<CartItem[]>([]);
  const [iframeHeights, setIframeHeights] = useState({
    [FRAME_NAMES.BOOK_LIST]: 500,
    [FRAME_NAMES.SINGLE_BOOK]: 500,
  });

  // Локално обрађене акције
  const handleAddToCart = useCallback((data) => {
    setCart(prevCart => {
      const existing = prevCart.find(item => item.bookId === data.payload.bookId);
      if (existing) {
        return prevCart.map(item =>
          item.bookId === data.payload.bookId
            ? { ...item, quantity: item.quantity + data.payload.quantity }
            : item
```

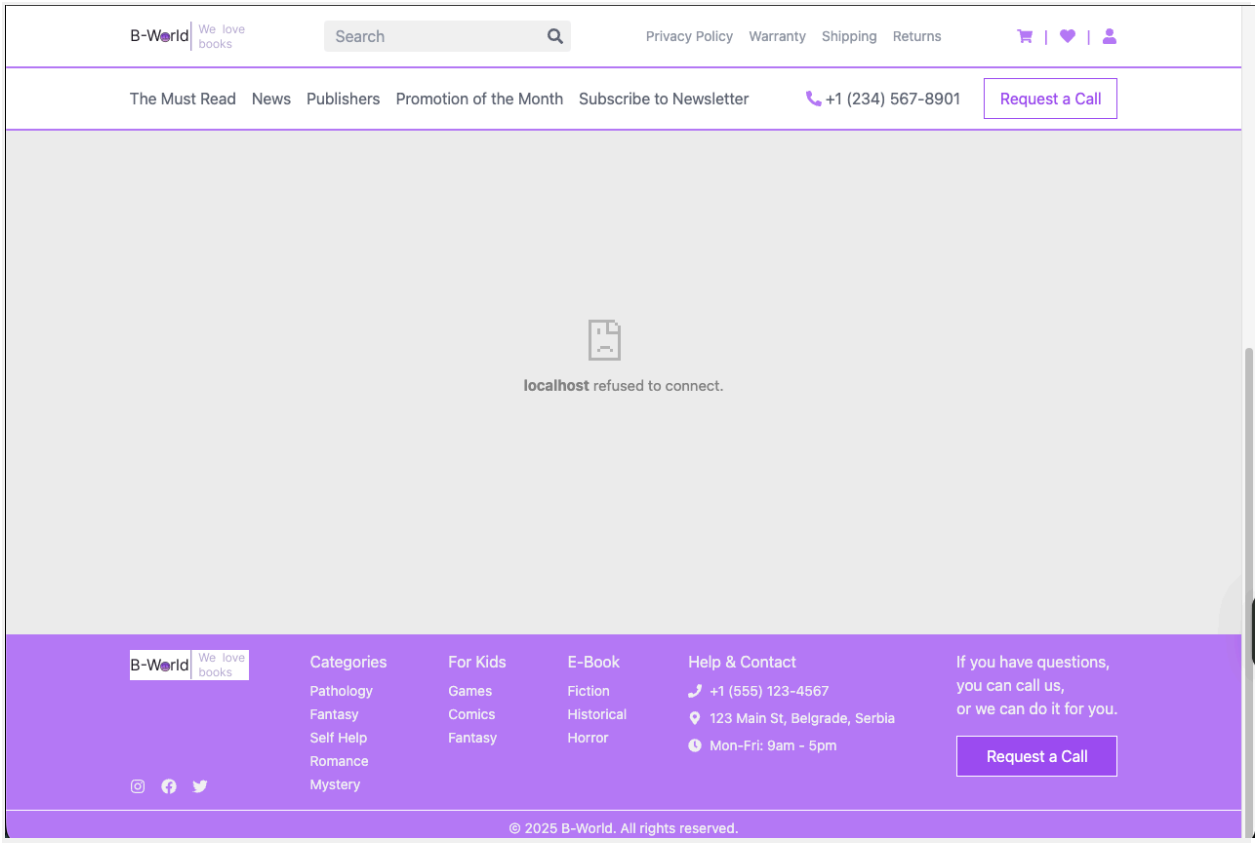
```
        );
      }
      return [...prevCart, data.payload];
    });
  }, []);

  // Регистрација слушаоца порука
  useEffect(() => {
    const unregister = registerMessageListener(MESSAGE_TYPE, (data) => {
      // Локална обрада
      if (data.action === MESSAGE_ACTIONS.ADD_TO_CART) {
        handleAddToCart(data);
        return;
      }
      if (data.action === MESSAGE_ACTIONS.RESIZE_IFRAME) {
        setIframeHeights(prev => ({
          ...prev,
          [data.payload.frameName]: data.payload.height,
        }));
        return;
      }

      // Прослеђивање одговарајућем iframe-у
      const config = IFRAME_ACTION_ROUTER[data.action];
      if (config) {
        const iframe = document.getElementById(config.frameName);
        if (iframe?.contentWindow) {
          sendMessage(iframe.contentWindow, config.origin, MESSAGE_TYPE, data);
        }
      }
    });
    return unregister;
  }, [handleAddToCart]);

  return (
    <div className="min-h-screen flex flex-col">
      <Header cart={cart} onSearch={handleSearch} />
      <main className="flex-1 container mx-auto px-4 py-6">
        <div className="grid grid-cols-1 lg:grid-cols-2 gap-6">
          <IframeErrorBoundary
            src={getAppOrigin('book-list')}
            id={FRAME_NAMES.BOOK_LIST}
            style={{ height: iframeHeights[FRAME_NAMES.BOOK_LIST] }}
          />
          <IframeErrorBoundary
            src={getAppOrigin('single-book')}
            id={FRAME_NAMES.SINGLE_BOOK}
            style={{ height: iframeHeights[FRAME_NAMES.SINGLE_BOOK] }}
          />
        </div>
      </main>
      <Footer />
    </div>
  );
}
```

Листинг 6.4: *Container App.tsx* - рутирање порука



Слика 6.3: *Container* апликација - *React*

6.1.4. *Book List* микрофронтенд (*Svelte*)

Book List микрофронтенд је имплементиран у *Svelte 5* радном оквиру, који се одликује компајлерским приступом и минималном додатном комплексношћу у времену извршавања. *Svelte* компонента се компајлира у оптимизовани *JavaScript* код који директно манипулише DOM-ом, без виртуелног DOM-а.

```
<!-- apps/book-list/src/App.svelte -->

<script lang="ts">
  import { onMount, afterUpdate } from 'svelte';
  import { registerMessageListener, sendMessage, getAppOrigin } from
'../../../shared/communication';
  import { MESSAGE_TYPE, MESSAGE_ACTIONS, FRAME_NAMES } from '../../../shared/types';
  import BookCard from './components/BookCard.svelte';

  let books = [];
  let loading = false;
  let error = null;

  async function getBooksFromGoogleApi(query) {
    if (!query.trim()) return;
    loading = true;
    error = null;

    try {
      const response = await fetch(

`https://www.googleapis.com/books/v1/volumes?q=${encodeURIComponent(query)}&maxResults=12`
      );
      if (!response.ok) throw new Error('API request failed');
      const data = await response.json();
      books = data.items || [];
    } catch (e) {
      error = 'Грешка при дохватању књига.';
```

```
    } finally {
      loading = false;
    }
  }

  function showSingleBook(event) {
    sendMessage(window.parent, getAppOrigin('container'), MESSAGE_TYPE, {
      action: MESSAGE_ACTIONS.SHOW_SINGLE_BOOK,
      payload: { bookId: event.detail.id }
    });
  }

  function addToCart(event) {
    const book = event.detail.book;
    sendMessage(window.parent, getAppOrigin('container'), MESSAGE_TYPE, {
      action: MESSAGE_ACTIONS.ADD_TO_CART,
      payload: {
        bookId: book.id,
        title: book.volumeInfo.title,
        authors: book.volumeInfo.authors,
        thumbnail: book.volumeInfo.imageLinks?.thumbnail,
        quantity: 1
      }
    });
  }

  onMount(() => {
    const unregister = registerMessageListener(MESSAGE_TYPE, (data) => {
      if (data.action === MESSAGE_ACTIONS.SEARCH_BOOK_LIST) {
        getBooksFromGoogleApi(data.payload.query);
      }
    });
    getBooksFromGoogleApi('javascript programming');
    return unregister;
  });

  afterUpdate(() => {
    sendMessage(window.parent, getAppOrigin('container'), MESSAGE_TYPE, {
      action: MESSAGE_ACTIONS.RESIZE_IFRAME,
      payload: { height: document.body.scrollHeight + 20, frameName: FRAME_NAMES.BOOK_LIST }
    });
  });
</script>

<div class="p-4">
  <h2 class="text-xl font-bold mb-4">Листа књига</h2>
  {#if loading}
    <div class="flex justify-center py-12">
      <div class="animate-spin rounded-full h-12 w-12 border-4 border-indigo-500"></div>
    </div>
  {:else if error}
    <div class="bg-red-100 text-red-700 p-4 rounded">{error}</div>
  {:else}
    <div class="grid grid-cols-1 sm:grid-cols-2 lg:grid-cols-3 gap-4">
      {#each books as book (book.id)}
        <BookCard {book} on:select={showSingleBook} on:addToCart={addToCart} />
      {/each}
    </div>
  {/if}
</div>
```

Листинг 6.5: *Book List App.svelte*



Слика 6.4: *Book List* микрофронтенд - *Svelte*

6.1.5. *Single Book* микрофронтенд (*Vue*)

Single Book микрофронтенд је имплементиран у *Vue 3* радном оквиру коришћењем *Options* API-ја. Овај микрофронтенд је одговоран за приказ детаљних информација о одабраној књизи и омогућава кориснику додавање књиге у корпу са одабраном количином.

```
<!-- apps/single-book/src/App.vue -->

<template>
  <div class="p-4">
    <h2 class="text-xl font-bold mb-4">Детаљи књиге</h2>

    <div v-if="loading" class="flex justify-center py-12">
      <div class="animate-spin rounded-full h-12 w-12 border-4 border-indigo-500"></div>
    </div>

    <div v-else-if="!book" class="text-gray-500 text-center py-12">
      Одаберите књигу из листе за приказ детаља.
    </div>

    <div v-else class="bg-white rounded-lg shadow-md p-6">
      <div class="flex flex-col md:flex-row gap-6">
        <img :src="coverImage" :alt="book.volumeInfo.title" class="w-48 rounded-lg shadow" />
        <div class="flex-1">
          <h3 class="text-2xl font-bold mb-2">{{ book.volumeInfo.title }}</h3>
          <p class="text-lg text-gray-600 mb-4">{{ authors }}</p>
          <p class="text-gray-700 mb-6">{{ sanitizedDescription }}</p>

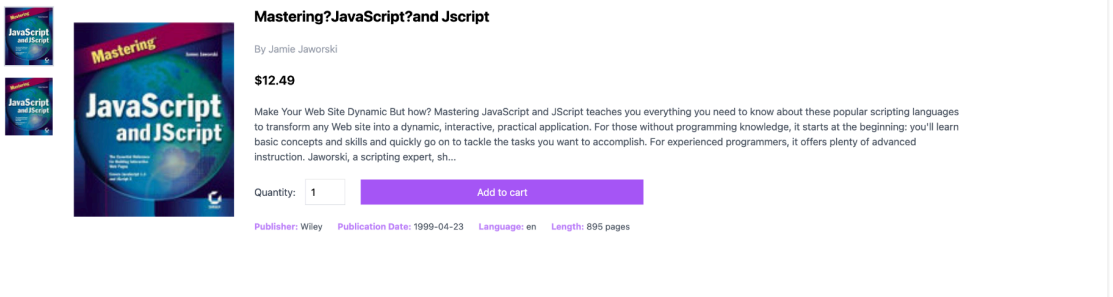
          <div class="flex items-center gap-4 pt-4 border-t">
            <span v-if="price" class="text-2xl font-bold text-indigo-600">
              {{ price.amount }} {{ price.currencyCode }}
            </span>
            <input type="number" v-model.number="quantity" min="1" max="10" class="w-16 border
rounded" />
            <button @click="addToCart" class="px-6 py-2 bg-indigo-600 text-white rounded-lg">
              🛒 Додај у корпу
            </button>
          </div>
        </div>
      </div>
    </div>
```

```
  </div>
</template>

<script>
import { registerMessageListener, sendMessage, getAppOrigin } from
'../../../shared/communication';
import { MESSAGE_TYPE, MESSAGE_ACTIONS, FRAME_NAMES } from '../../../shared/types';
import { stripHtml } from '../../../shared/sanitization';

export default {
  data() {
    return { book: null, loading: false, quantity: 1, unregisterListener: null };
  },
  computed: {
    authors() { return this.book?.volumeInfo.authors?.join(', ') || ''; },
    coverImage() { return this.book?.volumeInfo.imageLinks?.medium || '/placeholder.png'; },
    sanitizedDescription() { return stripHtml(this.book?.volumeInfo.description || ''); },
    price() { return this.book?.saleInfo?.listPrice; }
  },
  mounted() {
    this.unregisterListener = registerMessageListener(MESSAGE_TYPE, (data) => {
      if (data.action === MESSAGE_ACTIONS.SHOW_SINGLE_BOOK) {
        this.fetchBookDetails(data.payload.bookId);
      }
    });
  },
  beforeUnmount() { this.unregisterListener?.(); },
  methods: {
    async fetchBookDetails(bookId) {
      this.loading = true;
      const response = await fetch(`https://www.googleapis.com/books/v1/volumes/${bookId}`);
      this.book = await response.json();
      this.loading = false;
    },
    addToCart() {
      sendMessage(window.parent, getAppOrigin('container'), MESSAGE_TYPE, {
        action: MESSAGE_ACTIONS.ADD_TO_CART,
        payload: { bookId: this.book.id, title: this.book.volumeInfo.title, quantity:
this.quantity }
      });
    }
  }
};
</script>
```

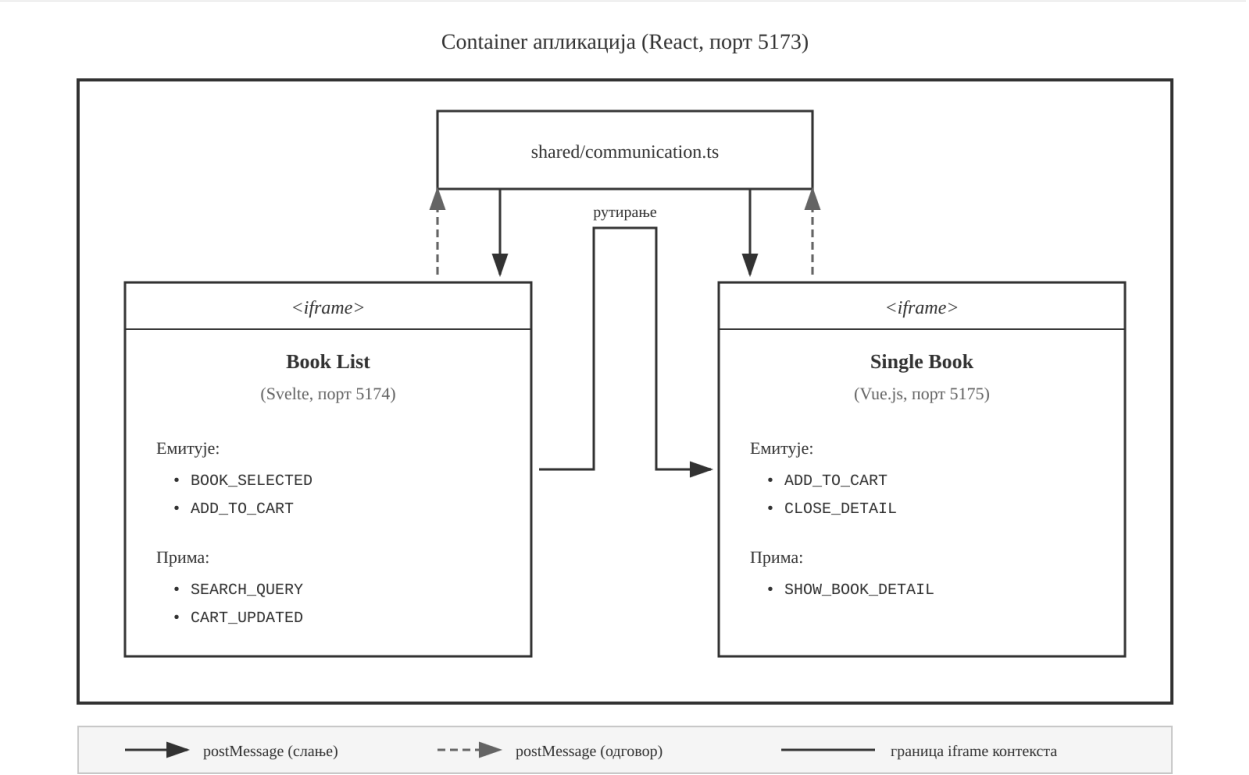
Листинг 6.6: *Single Book App.vue*



Слика 6.5: *Single Book* микрофронтенд - *Vue*

6.1.6. Дијаграм комуникације

Комуникација у *iframe* имплементацији слиједи *hub-and-spoke* образац гдје *Container* служи као централни чвор кроз који пролазе све поруке. Типичан ток при селекцији књиге: Корисник кликће на књигу у *Book List*; *Book List* шаље *SHOW_SINGLE_BOOK* родитељу; *Container* прослеђује поруку *Single Book iframe*-у; *Single Book* прима податке и приказује их.



Слика 6.6: Дијаграм тока комуникације у *iframe* приступу

Акција	Пошиљалац	Прималац	Payload
SEARCH_BOOK_LIST	*Container*	*Book List*	*{ query: string }*
SHOW_SINGLE_BOOK	*Book List*	*Single Book*	*{ bookId: string }*
ADD_TO_CART	*Single Book*	*Container*	*{ bookId, title, quantity }*
RESIZE_IFRAME	*Book List / Single Book*	*Container*	*{ height, frameName }*

Табела 6.1: Преглед порука у *iframe* комуникацији

6.2. Имплементација *Module Federation* приступа

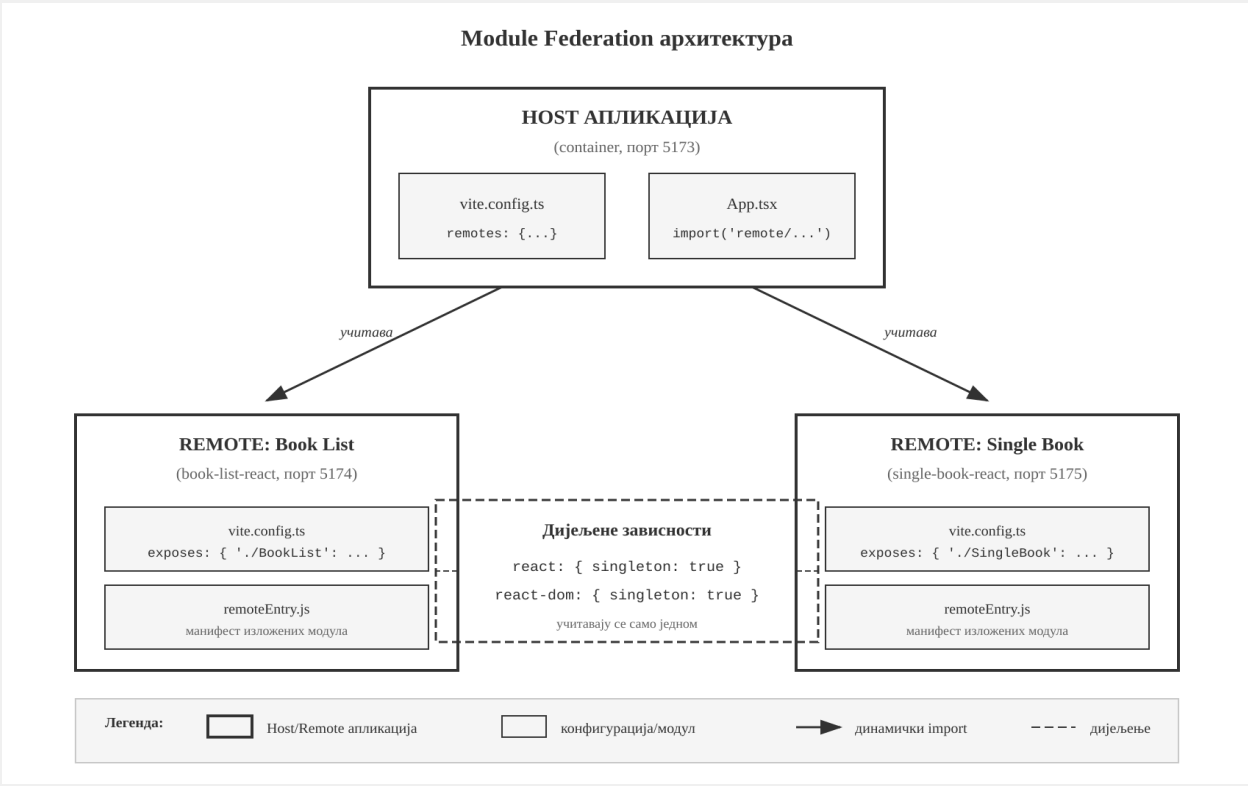
Module Federation имплементација демонстрира дијељење кода између независних апликација у току времена извршавања. За разлику од *iframe* приступа гдје је свака

апликација потпуно изолована, *Module Federation* омогућава дијељење *React* компоненти, контекста, и библиотека између *Host* и *Remote* апликација.

6.2.1. Структура пројекта

```
mf-micro-frontend/
├── host-react/              # Host апликација (порт 5001)
│   ├── src/
│   │   ├── context/
│   │   │   └── GlobalContext.tsx    # Дијељени контекст
│   │   ├── components/
│   │   ├── App.tsx
│   │   └── main.tsx
│   ├── package.json
│   └── vite.config.ts
│
├── book-list-react/         # Remote: листа књига (порт 5002)
│   ├── src/
│   │   └── App.tsx          # Exposed module
│   └── vite.config.ts
│
├── single-book-react/       # Remote: детаљи књиге (порт 5003)
│   ├── src/
│   │   └── App.tsx          # Exposed module
│   └── vite.config.ts
│
└── shared-react/            # Remote: дијељене компоненте (порт 5099)
    ├── src/
    │   └── components/
    │       ├── Button.tsx
    │       ├── Search.tsx
    │       └── ErrorBoundary.tsx
    └── vite.config.ts
```

Листинг 6.7: Структура *Module Federation* пројекта



Слика 6.7: Архитектура *Module Federation* са *Host* и *Remote* апликацијама

6.2.2. *Module Federation* Конфигурација

Конфигурација *Module Federation*-а врши се у *vite.config.ts* датотеци сваке апликације коришћењем *@originjs/vite-plugin-federation* додатка. *Host* апликација конфигурише листу *Remote* апликација и дијељених зависности.

```
// host-react/vite.config.ts

import { defineConfig, loadEnv } from 'vite';
import react from '@vitejs/plugin-react';
import federation from '@originjs/vite-plugin-federation';

export default defineConfig(({ mode }) => {
  const env = loadEnv(mode, process.cwd(), '');

  return {
    plugins: [
      react(),
      federation({
        name: 'host',
        exposes: {
          './GlobalContext': './src/context/GlobalContext.tsx',
        },
        remotes: {
          bookList: `${env.VITE_BOOK_LIST_APP_URL}/assets/remoteEntry.js`,
          singleBook: `${env.VITE_SINGLE_BOOK_APP_URL}/assets/remoteEntry.js`,
          shared: `${env.VITE_SHARED_APP_URL}/assets/remoteEntry.js`,
        },
        shared: {
          react: { singleton: true, requiredVersion: '^18.0.0' },
          'react-dom': { singleton: true, requiredVersion: '^18.0.0' },
        },
      }),
    ],
    server: { port: 5001, cors: true },
  };
});
```

Листинг 6.8: *Host vite.config.ts*

```
// book-list-react/vite.config.ts

federation({
  name: 'bookList',
  filename: 'remoteEntry.js',
  exposes: {
    './App': './src/App',
  },
  remotes: {
    host: `${env.VITE_HOST_APP_URL}/assets/remoteEntry.js`,
    shared: `${env.VITE_SHARED_APP_URL}/assets/remoteEntry.js`,
  },
  shared: {
    react: { singleton: true },
    'react-dom': { singleton: true },
  },
})
```

Листинг 6.9: *Remote vite.config.ts (book-list-react)*

6.2.3. Дијељени Глобални Контекст

Централни елемент комуникације у *Module Federation* имплементацији је *GlobalContext* - *React Context* који *Host* апликација креира и излаже, а *Remote* апликације конзумирају. Овај образац омогућава дијељење стања између свих микрофронтенда без експлицитне размјене порука.

```
// host-react/src/context/GlobalContext.tsx

import { createContext, useContext, useState, useCallback, ReactNode } from 'react';

export interface CartItem {
  bookId: string;
  title: string;
  authors?: string[];
  thumbnail?: string;
  price?: number;
  quantity: number;
}

export interface GlobalContextType {
  cart: CartItem[];
  selectedBook: string | null;
  searchText: string;
  setCart: React.Dispatch<React.SetStateAction<CartItem[]>>;
  setSelectedBook: React.Dispatch<React.SetStateAction<string | null>>;
  setSearchText: React.Dispatch<React.SetStateAction<string>>;
  addToCart: (item: CartItem) => void;
  removeFromCart: (bookId: string) => void;
}

const GlobalContext = createContext<GlobalContextType | undefined>(undefined);

export function GlobalContextProvider({ children }: { children: ReactNode }) {
  const [cart, setCart] = useState<CartItem[]>([]);
  const [selectedBook, setSelectedBook] = useState<string | null>(null);
  const [searchText, setSearchText] = useState<string>('');

  const addToCart = useCallback((item: CartItem) => {
    setCart(prev => {
      const existing = prev.findIndex(i => i.bookId === item.bookId);
      if (existing !== -1) {
        const updated = [...prev];
        updated[existing].quantity += item.quantity;
        return updated;
      }
      return [...prev, item];
    });
  }, []);

  const removeFromCart = useCallback((bookId: string) => {
    setCart(prev => prev.filter(item => item.bookId !== bookId));
  }, []);

  return (
    <GlobalContext.Provider value={{
      cart, selectedBook, searchText,
      setCart, setSelectedBook, setSearchText,
      addToCart, removeFromCart
    }}>
      {children}
    </GlobalContext.Provider>
  );
}

export function useGlobalContext() {
  const context = useContext(GlobalContext);
  if (!context) throw new Error('useGlobalContext must be within Provider');
  return context;
}
```

Листинг 6.10: *GlobalContext* дефиниција

6.2.4. *Host* апликација са динамичким учитавањем

Host апликација користи *React.lazy()* и *Suspense* за динамичко учитавање *Remote* компоненти. Код *Remote* апликације се преузима тек када је компонента потребна, смањујући иницијално вријеме учитавања.

```
// host-react/src/App.tsx

import { Suspense, lazy } from 'react';
import { GlobalContextProvider, useGlobalContext } from './context/GlobalContext';
import Header from './components/Header';

// Динамичко учитавање Remote компоненти
const BookList = lazy(() => import('bookList/App'));
const SingleBook = lazy(() => import('singleBook/App'));
import { ModuleFederationErrorBoundary } from 'shared/ModuleFederationErrorBoundary';

function LoadingSpinner() {
  return (
    <div className="flex justify-center py-12">
      <div className="animate-spin rounded-full h-12 w-12 border-4 border-indigo-500" />
      <span className="ml-3">Учитавање модула...</span>
    </div>
  );
}

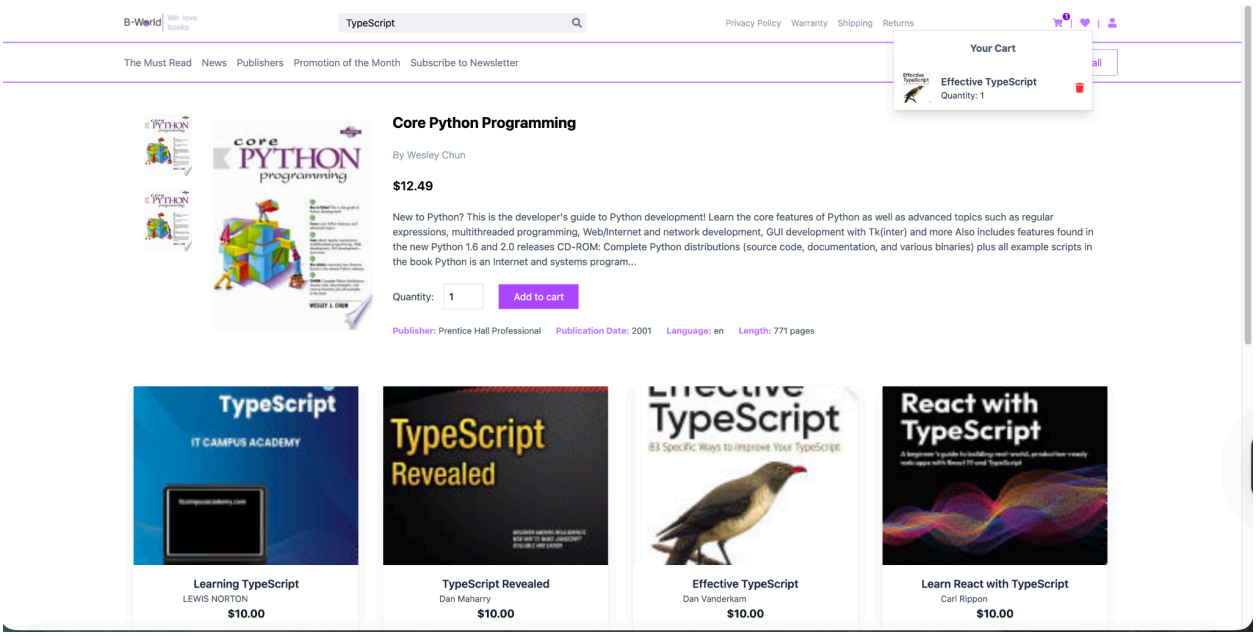
function AppContent() {
  const { selectedBook } = useGlobalContext();

  return (
    <div className="min-h-screen flex flex-col">
      <Header />
      <main className="flex-1 container mx-auto px-4 py-6">
        <div className="grid grid-cols-1 lg:grid-cols-2 gap-6">
          <section className="bg-white rounded-lg shadow-md p-4">
            <h2 className="text-xl font-bold mb-4">Листа књига</h2>
            <ModuleFederationErrorBoundary moduleName="Book List">
              <Suspense fallback={<LoadingSpinner />}>
                <BookList />
              </Suspense>
            </ModuleFederationErrorBoundary>
          </section>

          <section className="bg-white rounded-lg shadow-md p-4">
            <h2 className="text-xl font-bold mb-4">Детаљи књиге</h2>
            <ModuleFederationErrorBoundary moduleName="Single Book">
              <Suspense fallback={<LoadingSpinner />}>
                {selectedBook ? <SingleBook /> : <p>Одаберите књигу...</p>}
              </Suspense>
            </ModuleFederationErrorBoundary>
          </section>
        </div>
      </main>
    </div>
  );
}

export default function App() {
  return (
    <GlobalContextProvider>
      <AppContent />
    </GlobalContextProvider>
  );
}
```

Листинг 6.11: *Host App.tsx*



Слика 6.8: *Module Federation Host* са учитаним *Remote* модулима

6.2.5. *Book List Remote* модул

Remote апликације увозе *useGlobalContext* hook директно од *Host* апликације коришћењем *Module Federation import* синтаксе. Ово омогућава транспарентан приступ глобалном стању.

```
// book-list-react/src/App.tsx

import { useState, useEffect } from 'react';
// Увоз контекста од Host апликације (Module Federation)
import { useGlobalContext } from 'host/GlobalContext';
import { ErrorBoundary } from 'shared/ErrorBoundary';
import BookCard from './components/BookCard';

export default function BookListApp() {
  const [books, setBooks] = useState([]);
  const [loading, setLoading] = useState(false);

  // Директан приступ глобалном стању
  const { searchText, setSelectedBook, addToCart } = useGlobalContext();

  useEffect(() => {
    if (!searchText.trim()) return;

    const fetchBooks = async () => {
      setLoading(true);
      const response = await fetch(
        `https://www.googleapis.com/books/v1/volumes?q=${encodeURIComponent(searchText)}`
      );
      const data = await response.json();
      setBooks(data.items || []);
      setLoading(false);
    };
    fetchBooks();
  }, [searchText]);

  // Директно ажурирање глобалног стања
  const handleSelectBook = (bookId) => setSelectedBook(bookId);

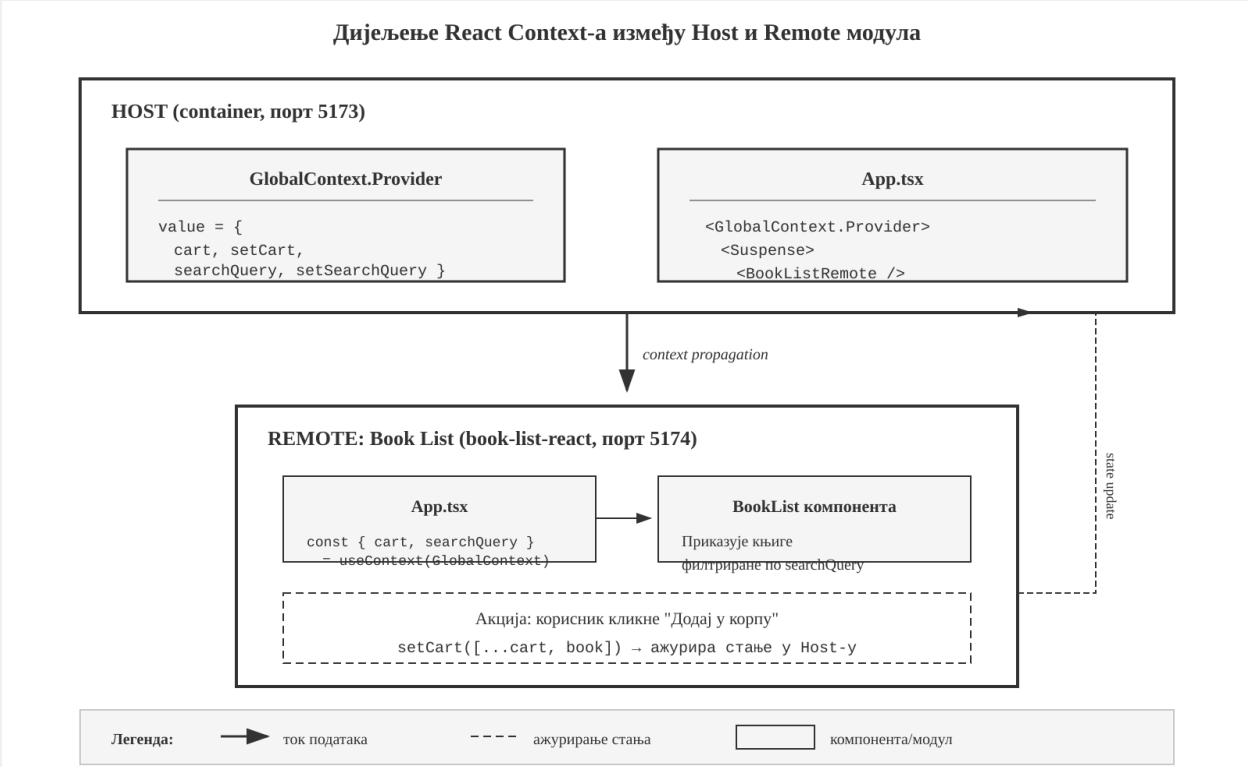
  const handleAddToCart = (book) => {
    addToCart({
      bookId: book.id,
      title: book.volumeInfo.title,
```

```
      quantity: 1
    });
  };

  if (loading) return <LoadingSpinner />;

  return (
    <ErrorBoundary>
      <div className="grid grid-cols-1 sm:grid-cols-2 lg:grid-cols-3 gap-4">
        {books.map(book => (
          <BookCard
            key={book.id}
            book={book}
            onSelect={() => handleSelectBook(book.id)}
            onAddToCart={() => handleAddToCart(book)}
          />
        ))}
      </div>
    </ErrorBoundary>
  );
}
```

Листинг 6.12: *Book List Remote App.tsx*



Слика 6.9: *Book List Remote* са дијељеним контекстом

Апликација	Изложени модули	Конзумира од
host-react	*./GlobalContext*	*bookList, singleBook, shared*
book-list-react	*./App*	*host, shared*
single-book-react	*./App*	*host, shared*
shared-react	*./Button, ./Search, ./ErrorBoundary*	-

Табела 6.2: *Module Federation* изложени модули

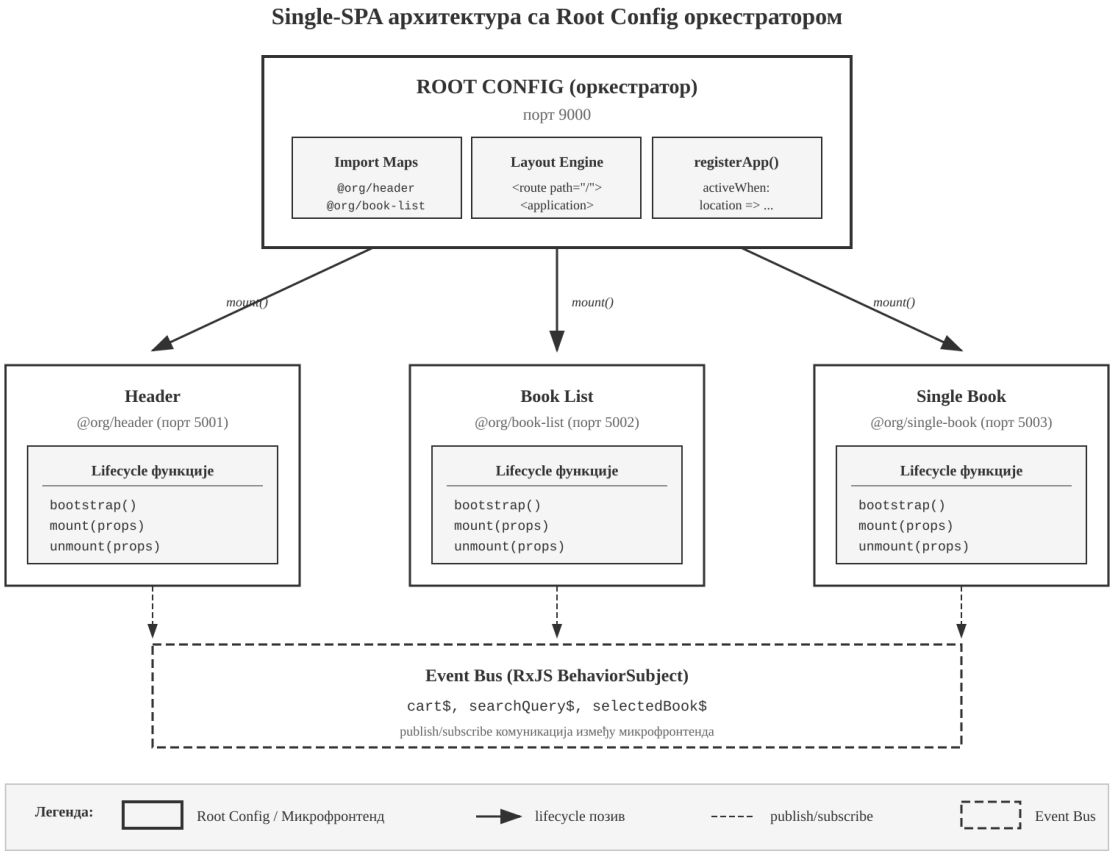
6.3. Имплементација *Single-SPA* приступа

Single-SPA имплементација демонстрира агностички приступ радним оквирима микрофронтенд архитектуре гдје централни оркестратор (*root-config*) управља животним циклусом независних *JavaScript* апликација. За разлику од *Module Federation*-а који се фокусира на дијељење модула, *Single-SPA* се фокусира на координацију када се апликације учитавају, монтирају и демонтирају(*bootstrap*, *mount*, *unmount*).

6.3.1. Структура пројекта

```
single-spa-micro-frontends/
├── root-config/              # Оркестратор (порт 3005)
│   ├── src/
│   │   ├── microfrontend-layout.html
│   │   └── main.ts
│   └── index.html            # Import Maps
│
├── event-bus/                # RxJS комуникација (порт 5173)
│   ├── src/
│   │   ├── eventBus.ts       # BehaviorSubject стања
│   │   └── main.ts
│   └── package.json
│
├── header/                   # Header (порт 3008)
├── footer/                   # Footer (порт 3009)
├── book-list/                # Book List (порт 3006)
└── single-book/              # Single Book (порт 3007)
```

Листинг 6.13: Структура *Single-SPA* пројекта



Слика 6.10: Архитектура *Single-SPA* са *Root Config* оркестратором

6.3.2 *Root Config* и *Import Maps*

Root Config је централна тачка *Single-SPA* апликације. *Import Maps* у *index.html* дефинишу мапирање логичких имена модула на стварне URL-ове, омогућавајући дијељење библиотека (*React*, *single-spa*) између свих микрофронтенда.

```
<!-- root-config/index.html -->

<script type="importmap">
  {
    "imports": {
      "@example/root-config": "http://localhost:3005/src/main.ts",
      "@single-spa/book-list": "http://localhost:3006/src/main.ts",
      "@single-spa/single-book": "http://localhost:3007/src/main.ts",
      "@single-spa/header": "http://localhost:3008/src/main.ts",
      "@single-spa/footer": "http://localhost:3009/src/main.ts",
      "@single-spa/eventBus": "http://localhost:5173/src/main.ts",

      "react": "https://ga.jspm.io/npm:react@18.2.0/dev.index.js",
      "react-dom": "https://ga.jspm.io/npm:react-dom@18.2.0/dev.index.js",
      "single-spa": "https://ga.jspm.io/npm:single-spa@5.9.5/lib/esm/single-spa.min.js"
    }
  }
</script>
```

Листинг 6.14: *Import Maps* конфигурација

```
<!-- root-config/src/microfrontend-layout.html -->

<single-spa-router>
  <main class="min-h-screen flex flex-col">
    <route default>
      <application name="@single-spa/eventBus"></application>
      <application name="@single-spa/header"></application>

      <div class="flex-1 container mx-auto px-4 py-6">
        <div class="grid grid-cols-1 lg:grid-cols-2 gap-6">
          <section class="bg-white rounded-lg shadow-md p-4">
            <application name="@single-spa/book-list"></application>
          </section>
          <section class="bg-white rounded-lg shadow-md p-4">
            <application name="@single-spa/single-book"></application>
          </section>
        </div>
      </div>

      <application name="@single-spa/footer"></application>
    </route>
  </main>
</single-spa-router>
```

Листинг 6.15: Микрофронтенд *Layout*

```
// root-config/src/main.ts

import { registerApplication, start } from 'single-spa';
import { constructApplications, constructRoutes, constructLayoutEngine } from
'single-spa-layout';
import microfrontendLayout from './microfrontend-layout.html?raw';

const routes = constructRoutes(microfrontendLayout);
const applications = constructApplications({
  routes,
  loadApp({ name }) {
    return import(/* @vite-ignore */ name);
  },
});
```

```
const layoutEngine = constructLayoutEngine({ routes, applications });

applications.forEach(registerApplication);
layoutEngine.activate();
start({ urlRerouteOnly: true });
```

Листинг 6.16: *Root Config main.ts*

6.3.3. *Event Bus* - RxJS комуникација

Event Bus микрофронтенд је срце комуникације у *Single-SPA* имплементацији. Користи *RxJS BehaviorSubject* за чување и емитовање стања. *BehaviorSubject* чува тренутну вриједност и емитује је сваком новом претплатнику.

```
// event-bus/src/eventBus.ts

import { BehaviorSubject } from 'rxjs';

export interface CartItem {
  bookId: string;
  title: string;
  quantity: number;
}

// BehaviorSubject стања
export const cartState$ = new BehaviorSubject<{ cart: CartItem[] }>({ cart: [] });
export const searchState$ = new BehaviorSubject<{ query: string }>({ query: '' });
export const selectedBookState$ = new BehaviorSubject<{ book: string | null }>({ book: null
});

// Акције
export function setSelectedBook(bookId: string | null) {
  selectedBookState$.next({ book: bookId });
}

export function setSearchQuery(query: string) {
  searchState$.next({ query });
}

export function addToCart(item: CartItem) {
  const current = cartState$.getValue();
  const existing = current.cart.findIndex(i => i.bookId === item.bookId);

  if (existing !== -1) {
    const updated = [...current.cart];
    updated[existing].quantity += item.quantity;
    cartState$.next({ cart: updated });
  } else {
    cartState$.next({ cart: [...current.cart, item] });
  }
}

export function removeFromCart(bookId: string) {
  const updated = cartState$.getValue().cart.filter(i => i.bookId !== bookId);
  cartState$.next({ cart: updated });
}
```

Листинг 6.17: *Event Bus* имплементација

```
// event-bus/src/main.ts

import * as eventBus from './eventBus';

declare global {
  interface Window { eventBus: typeof eventBus; }
}
```

```
export function bootstrap() { return Promise.resolve(); }

export function mount() {
  window.eventBus = eventBus;  // Глобално излагање
  return Promise.resolve();
}

export function unmount() {
  delete window.eventBus;
  return Promise.resolve();
}

mount();  // Аутоматски mount
```

Листинг 6.18: *Event Bus* животни циклус и глобално излагање

6.3.4. *React* микрофронтенд интеграција

React микрофронтенди користе *single-spa-react* помоћну библиотеку која аутоматски генерише функције животног циклуса (*bootstrap*, *mount*, *unmount*) из *React* компоненте.

```
// book-list/src/main.ts

import React from 'react';
import ReactDOMClient from 'react-dom/client';
import singleSpaReact from 'single-spa-react';
import App from './App';

const lifecycles = singleSpaReact({
  React,
  ReactDOMClient,
  rootComponent: App,
  errorBoundary(err) {
    return React.createElement('div', { className: 'text-red-600' },
      'Грешка у Book List: ' + err.message
    );
  },
});

export const { bootstrap, mount, unmount } = lifecycles;
```

Листинг 6.19: *Book List main.ts*

```
// book-list/src/App.tsx

import { useState, useEffect } from 'react';

export default function BookListApp() {
  const [books, setBooks] = useState([]);
  const [searchText, setSearchText] = useState('');

  // Претплата на searchState$ Observable
  useEffect(() => {
    const eventBus = window.eventBus;
    if (!eventBus) return;

    const subscription = eventBus.searchState$.subscribe((state) => {
      setSearchText(state.query);
    });

    return () => subscription.unsubscribe();
  }, []);

  // Дохватање књига при промјени searchText
  useEffect(() => {
    if (!searchText.trim()) {
      fetchBooks('javascript');
      return;
```

```
    }
    fetchBooks(searchText);
  }, [searchText]);

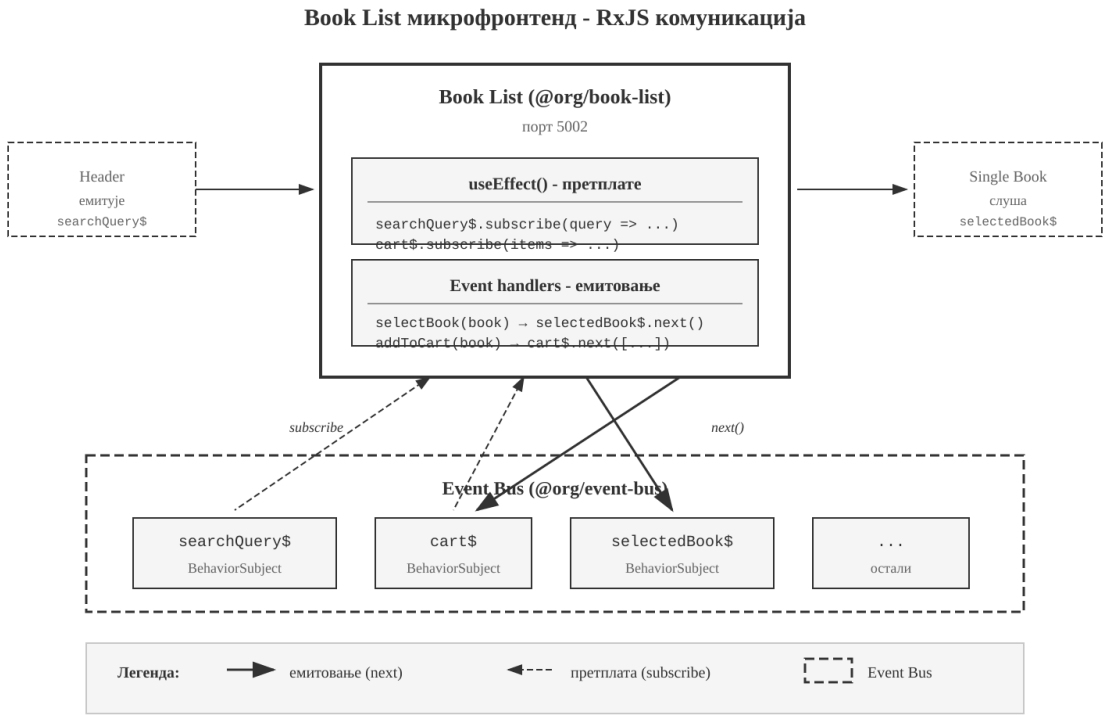
  async function fetchBooks(query) {
    const response = await fetch(
      `https://www.googleapis.com/books/v1/volumes?q=${query}`
    );
    const data = await response.json();
    setBooks(data.items || []);
  }

  // Емитовање догађаја преко EventBus-a
  function handleSelectBook(bookId) {
    window.eventBus?.setSelectedBook(bookId);
  }

  function handleAddToCart(book) {
    window.eventBus?.addToCart({
      bookId: book.id,
      title: book.volumeInfo.title,
      quantity: 1
    });
  }

  return (
    <div className="grid grid-cols-3 gap-4">
      {books.map(book => (
        <BookCard
          key={book.id}
          book={book}
          onSelect={() => handleSelectBook(book.id)}
          onAddToCart={() => handleAddToCart(book)}
        />
      ))}
    </div>
  );
}
```

Листинг 6.20: *Book List App.tsx* са *RxJS* претплатом



Слика 6.11: *Single-SPA Book List* са *RxJS* комуникацијом

6.3.5. Header микрофронтенд

```
// header/src/App.tsx

import { useState, useEffect } from 'react';

export default function HeaderApp() {
  const [cart, setCart] = useState([]);
  const [searchQuery, setSearchQuery] = useState('');

  // Претплата на стање корпе
  useEffect(() => {
    const eventBus = window.eventBus;
    if (!eventBus) return;

    const subscription = eventBus.cartState$.subscribe((state) => {
      setCart(state.cart);
    });

    return () => subscription.unsubscribe();
  }, []);

  function handleSearch(e) {
    e.preventDefault();
    if (searchQuery.trim()) {
      window.eventBus?.setSearchQuery(searchQuery.trim());
    }
  }

  function handleRemoveFromCart(bookId) {
    window.eventBus?.removeFromCart(bookId);
  }

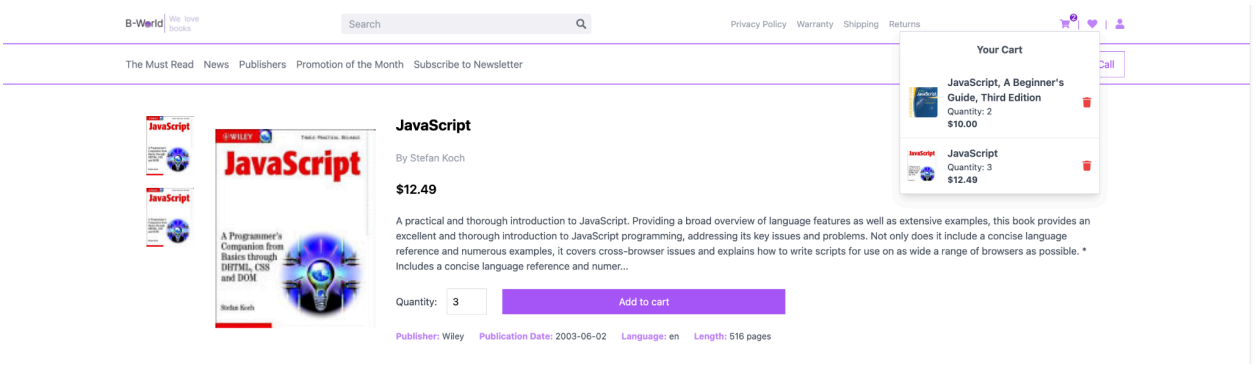
  const totalItems = cart.reduce((sum, item) => sum + item.quantity, 0);

  return (
    <header className="bg-indigo-600 text-white p-4">
      <div className="container mx-auto flex justify-between items-center">
        <h1 className="text-2xl font-bold">B-World</h1>

        <form onSubmit={handleSearch} className="flex-1 max-w-xl mx-8">
          <input
            type="text"
            value={searchQuery}
            onChange={(e) => setSearchQuery(e.target.value)}
            placeholder="Претражи књиге..."
            className="w-full px-4 py-2 rounded text-gray-900"
          />
        </form>

        <button className="relative px-4 py-2 bg-indigo-700 rounded">
          {totalItems > 0 && (
            <span className="absolute -top-2 -right-2 bg-red-500 rounded-full w-5 h-5
text-xs">
              {totalItems}
            </span>
          )}
        </button>
      </div>
    </header>
  );
}
```

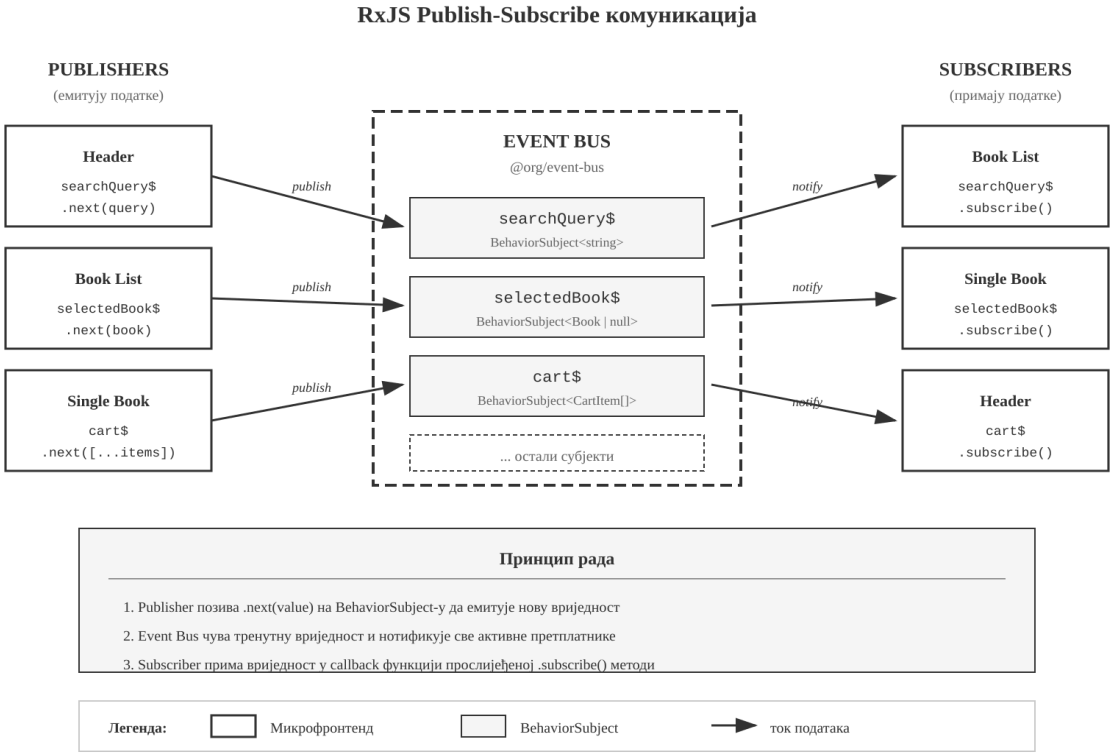
Листинг 6.21: *Header App.tsx*



Слика 6.12: *Single-SPA Header* са корпом

6.3.6. Дијаграм RxJS комуникације

Комуникација у *Single-SPA* имплементацији слиједи *publish-subscribe* образац. EventBus служи као централно складиште стања, а микрофронтенди се претплаћују на релевантне *Observable-e*. Промјена стања у једном микрофронтенду аутоматски се пропагира свим претплатницима.



Слика 6.13: Дијаграм RxJS *publish-subscribe* комуникације

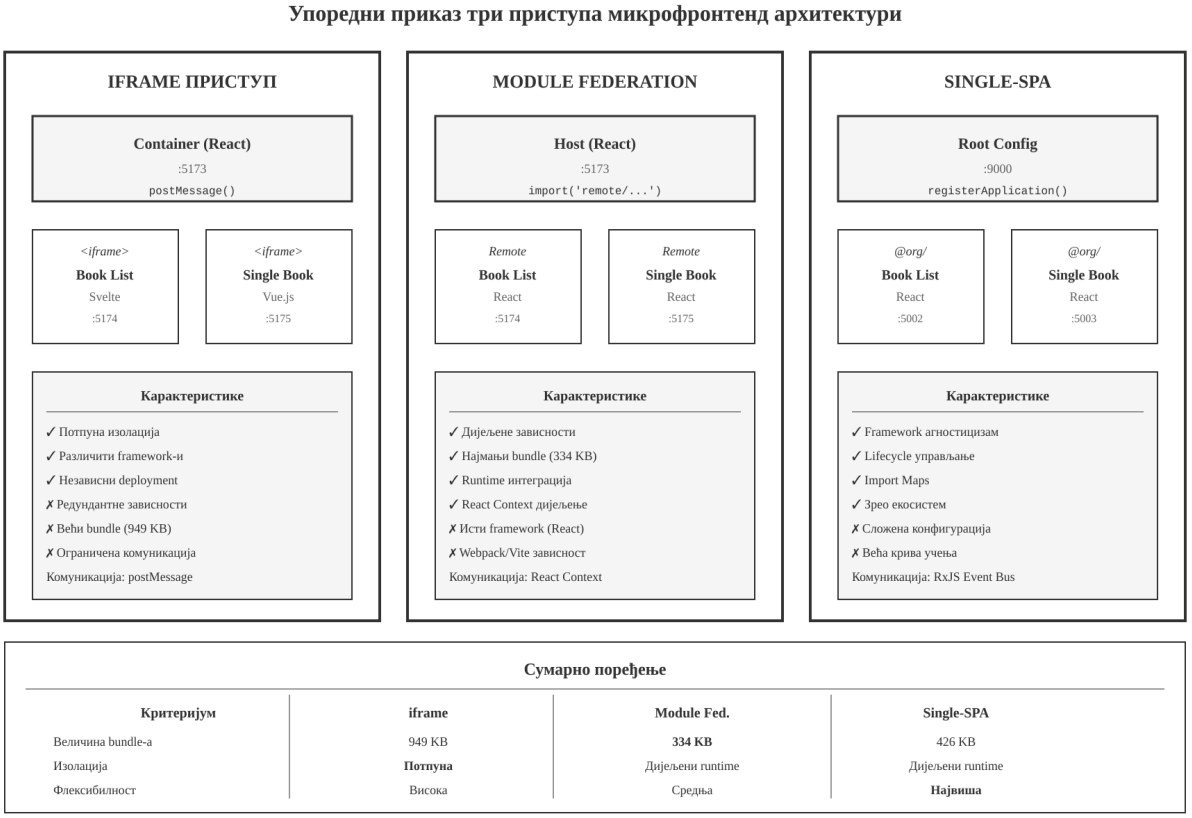
Observable	Тип	Претплатници
cartState$	*{ cart: CartItem[] }*	*Header, Single Book*
searchState$	*{ query: string }*	*Book List*
selectedBookState$	*{ book: string | null }*	*Single Book*

Табела 6.3: *RxJS Observable* стања

6.4. Завршне напомене о имплементацији

Све три имплементације успјешно реализују идентичне функционалности *B-World e-commerce* апликације, демонстрирајући да исти кориснички захтјеви могу бити испуњени кроз фундаментално различите архитектонске приступе. Свака имплементација има своје специфичности у погледу структуре кода, комуникационих образаца, и комплексности конфигурације.

Iframe приступ показује највећу технолошку разноликост (*React*, *Svelte*, *Vue*) уз цијену сложеније комуникације путем *postMessage*-а. *Module Federation* нуди најдубљу интеграцију кроз дијељени *React Context*, али ограничава избор радног оквира. *Single-SPA* пружа баланс између флексибилности и структуре, уз моћан реактивни комуникациони механизам путем RxJS-а.



Слика 6.14: Упоредни приказ сва три приступа

У наредном поглављу представљена је детаљна компаративна анализа сва три приступа на основу мјерљивих критеријума као што су перформансе, величина *bundle*-а, и сложеност имплементације.

7. Компаративна Анализа

У овом поглављу представљена је свеобухватна компаративна анализа три имплементирана приступа микрофронтенд архитектури. Анализа је спроведена на основу објективних критеријума који су релевантни за доношење одлуке о избору приступа у реалним пројектима. Критеријуми укључују перформансе, сложеност имплементације, искуство програмера (*developer experience*), скалабилност, одржавање, и погодност за различите сценарије употребе.

Мјерења перформанси су спроведена у контролисаним условима коришћењем *Chrome DevTools* и *Lighthouse* алата. Сва три приступа су тестирана на истој хардверској конфигурацији, са симулираним мрежним условима који одражавају типично корисничко искуство.

7.1. Критеријуми евалуације

Евалуација микрофронтенд приступа захтијева мултидимензионални приступ који узима у обзир техничке, организационе и пословне факторе. Сљедећи критеријуми су идентификовани као кључни за упоредну анализу:

Перформансе обухватају вријеме иницијалног учитавања (*Time to First Contentful Paint*), вријеме до интерактивности (*Time to Interactive*), величину преузетих ресурса, и ефикасност кеширања. Ови показатељи директно утичу на корисничко искуство и SEO рангирање.

Сложеност имплементације мјери се кроз број конфигурационих датотека, линија кода потребних за интеграцију, и криву учења за програмере. Висока сложеност повећава ризик од грешака и успорава развој нових функционалности.

Искуство програмера (*Developer Experience*) укључује брзину *development server*-а, квалитет порука о грешкама, подршку за дебаговање, и доступност документације. Добро искуство програмера повећава продуктивност и смањује фрустрацију тима.

Скалабилност процјењује способност архитектуре да подржи раст броја микрофронтенда, тимова, и корисника без значајне деградације перформанси или повећања сложености. Одржавање анализира лакоћу ажурирања зависности, управљање верзијама, и изолацију грешака. Добра одрживост смањује техничко дуговање и укупне трошкове власништва (*Total Cost of Ownership*).

Категорија	Критеријум	Метрика
Перформансе	*First Contentful Paint*	Вријеме у милисекундама
Перформансе	*Time to Interactive*	Вријеме у милисекундама
Перформансе	Укупна величина *bundle*-а	Килобајти (KB)
Сложеност	Број конфигурационих фајлова	Број датотека
Сложеност	Линије интеграционог кода	LOC (*Lines of Code*)
DX	*Hot Module Replacement*	Вријеме у милисекундама
DX	Вријеме билдања	Секунде
Скалабилност	Независност *deployment*-а	Да/Не/Дјелимично
Одржавање	Изолација грешака	Потпуна/Дјелимична/Нема

Табела 7.1: Преглед критеријума евалуације

7.2. Анализа перформанси

Перформансе су мјерене коришћењем *Lighthouse* алата у *Chrome* прегледачу са симулираним мобилним условима (*throttled CPU* и мрежа). Мјерења су поновљена пет пута за сваки приступ, а резултати представљају просјечне вриједности. Тестирање је спроведено на продукционим *build*-овима свих апликација.

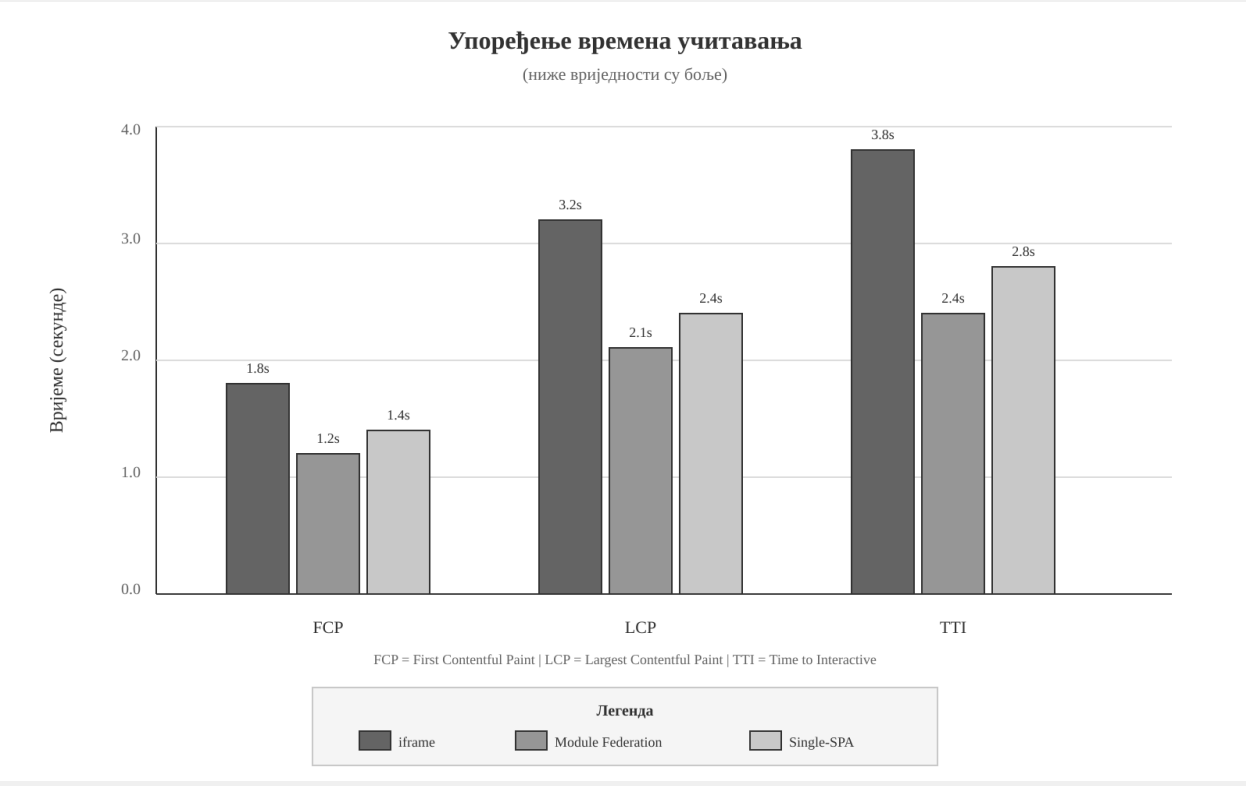
7.2.1. Вријеме учитавања

First Contentful Paint (FCP) мјери вријеме од почетка навигације до момента када прегледач рендерује први дио садржаја. Ова метрика је важна јер представља први визуелни одговор (*feedback*) кориснику да се страница учитава.

Метрика	iframe	Module Federation	Single-SPA
First Contentful Paint	1.8s	1.2s	1.4s
Largest Contentful Paint	3.2s	2.1s	2.4s
Time to Interactive	3.8s	2.4s	2.8s
Total Blocking Time	320ms	180ms	220ms
Cumulative Layout Shift	0.12	0.05	0.08

Табела 7.2: Резултати мјерења времена учитавања

Резултати показују да *iframe* приступ има најлошије перформансе учитавања, што је очекивано с обзиром на то да сваки *iframe* креира нови *browsing context* и учитава засебан документ. *Module Federation* показује најбоље резултате захваљујући ефикасном дијељењу зависности и *lazy loading*-у модула. *Single-SPA* се налази између, са незнатно лошијим резултатима од *Module Federation*-а због додатног *overhead*-а оркестратора.



Слика 7.1: Графикон упоређења времена учитавања (FCP, LCP, TTI)

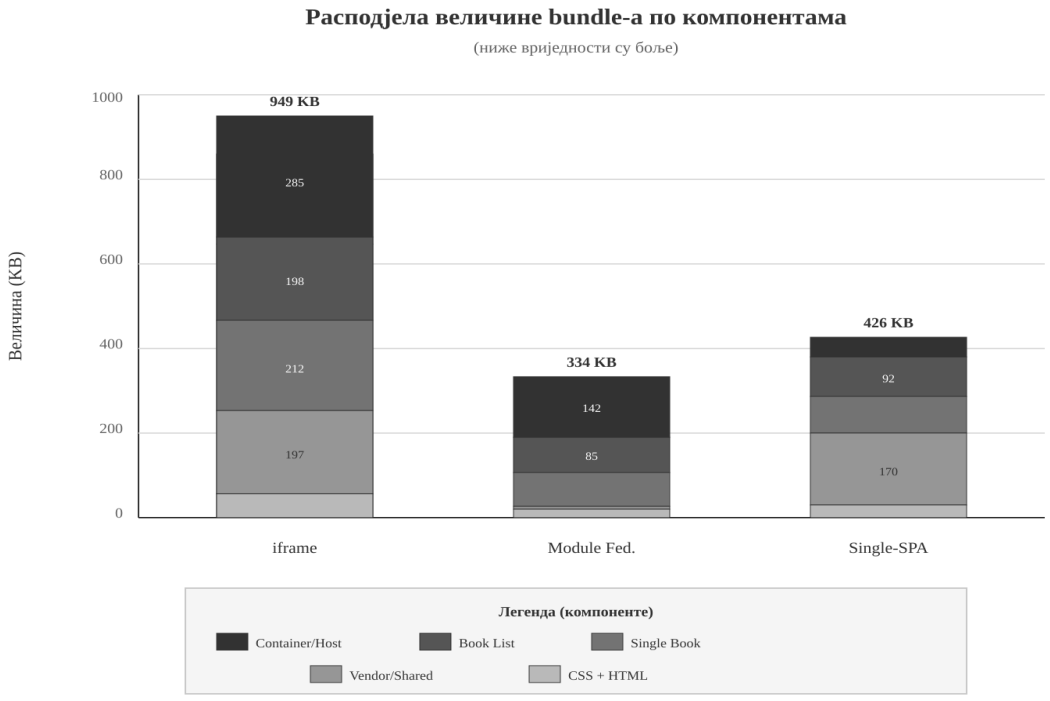
7.2.2. Величина bundle-а

Величина преузетих ресурса директно утиче на вријеме учитавања, посебно на мобилним мрежама. Анализа укључује *JavaScript*, CSS, и HTML датотеке, као и *vendor* библиотеке.

Ресурс	*iframe*	*Module Federation*	*Single-SPA*
JavaScript (укупно)	892 KB	312 KB	398 KB
Container/Host	285 KB	142 KB	48 KB
Book List	198 KB	85 KB	92 KB
Single Book	212 KB	78 KB	88 KB
Vendor/Shared	197 KB (x3)	7 KB	170 KB
CSS (укупно)	45 KB	18 KB	22 KB
HTML	12 KB	4 KB	6 KB
УКУПНО	**949 KB**	**334 KB**	**426 KB**

Табела 7.3: Величина bundle-а по приступу

Iframe приступ показује највећу укупну величину (949 KB) због тројног учитавања *vendor* библиотека. *React*, *React-DOM*, и друге зависности се учитавају засебно за сваки *iframe*, што резултира значајном редундантношћу. *Module Federation* постиже најмању величину (334 KB) захваљујући дијељењу *React-a* и других библиотека између свих модула у вријеме извршавања. *Single-SPA* је у средини (426 KB), гдје се дијељење постиже путем *Import Maps* и *CDN-hosted* библиотека.



Слика 7.2: Графикон расподјеле величине *bundle*-а по компонентама

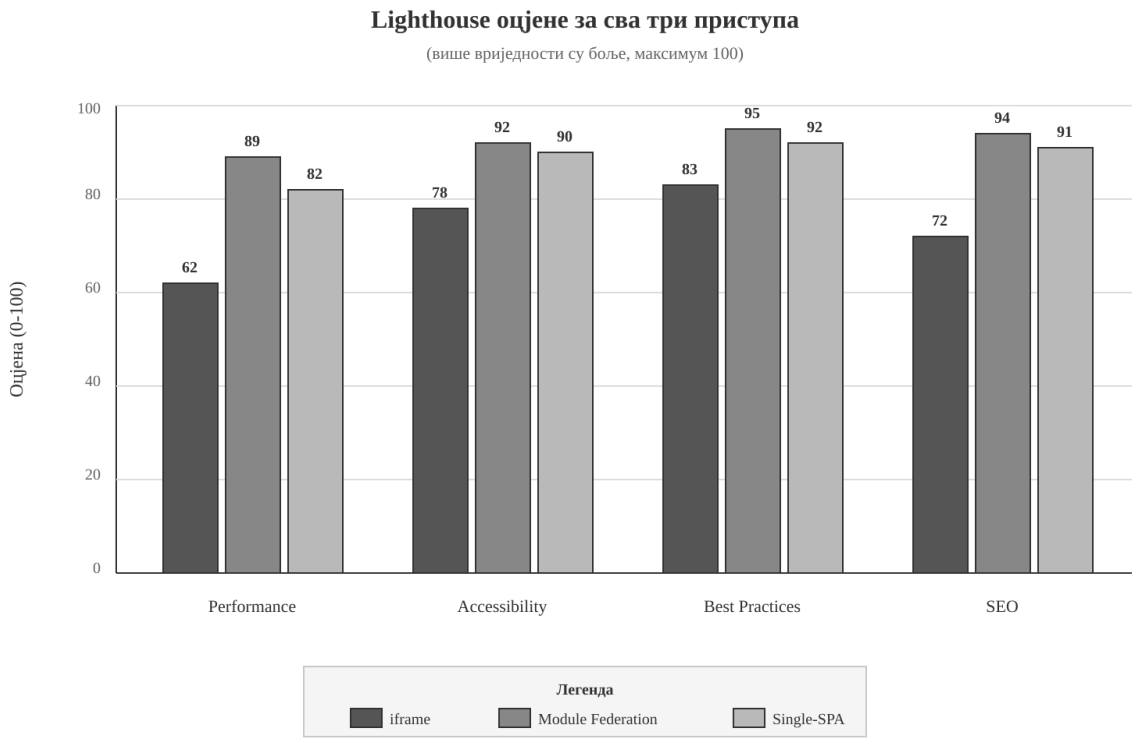
7.2.3 *Lighthouse* оцјене

Lighthouse је аутоматизовани алат за побољшање квалитета веб страница. Пружа оцјене у категоријама перформанси, приступачности, најбољих пракси, и SEO оптимизације.

Категорија	*iframe*	Module *Federation*	Single-SPA
Перформансе	62	89	82
Приступачност	78	92	90
Најбоље праксе	83	95	92
SEO	72	94	91

Табела 7.4: *Lighthouse* оцјене

Iframe приступ показује најниже оцјене у свим категоријама. Посебно је проблематична SEO оцјена (72) због тога што садржај унутар *iframe*-ова није директно видљив претраживачима. Оцјена приступачности (*accessibility)* (78) је такође нижа због потешкоћа са фокус менаџментом и читачем екрана навигацијом кроз вишеструке *iframe*-ове.



Слика 7.3: Radar графикон *Lighthouse* оцјена за сва три приступа

7.3. Анализа сложености имплементације

Сложеност имплементације је критичан фактор који утиче на вријеме развоја, ризик од грешака, и трошкове одржавања. Анализа укључује квантитативне метрике (број датотека, линије кода) и квалитативну процјену криве учења.

7.3.1. Конфигурациона сложеност

Метрика	*iframe*	*Module Federation*	*Single-SPA*
Број *vite.config* датотека	3	4	6
Број *package.json* датотека	4	4	6
Специфичне конфиг. датотеке	1 (*workspace*)	0	2 (*layout*, *importmap*)
Линије конфигурације	~120 LOC	~180 LOC	~220 LOC
Број екстерних зависности	3	5	8

Табела 7.5: Конфигурациона сложеност

Iframe приступ има најнижу конфигурациону сложеност захваљујући организацији *pnpm* радног окружења која елиминише потребу за сложеним конфигурацијама дијељења кода. *Module Federation* захтијева детаљну конфигурацију *exposes/remotes/shared* објеката у свакој апликацији. *Single-SPA* има највише конфигурационих датотека због потребе за *layout* дефиницијом, *Import Maps*, и засебним модулима животног циклуса.

7.3.2. Интеграциони код

Интеграциони код обухвата све линије кода које су специфичне за микрофронтенд архитектуру, искључујући пословну логику која би постојала и у монолитној апликацији.

Компонента	iframe	Module Federation	Single-SPA
Комуникациони слој	~150 LOC	~80 LOC	~120 LOC
Рутирање/оркестрација	~40 LOC	~30 LOC	~60 LOC
Управљање грешкама	~60 LOC	~50 LOC	~45 LOC
Дефинисање типова	~70 LOC	~90 LOC	~85 LOC
УКУПНО	**~320 LOC**	**~250 LOC**	**~310 LOC**

Табела 7.6: Обим интеграционог кода

Iframe приступ захтијева највише кода за комуникациони слој (*postMessage* апстракција, валидација поријекла, рутирање порука) али најмање за рутирање. *Module Federation* има најмањи укупни интеграциони код захваљујући *React Context*-у који елиминише потребу за експлицитном комуникацијом. *Single-SPA* захтијева значајан код за оркестрацију (функције животног циклуса, *layout* engine) али има добро документоване библиотеке које апстрахују сложеност.

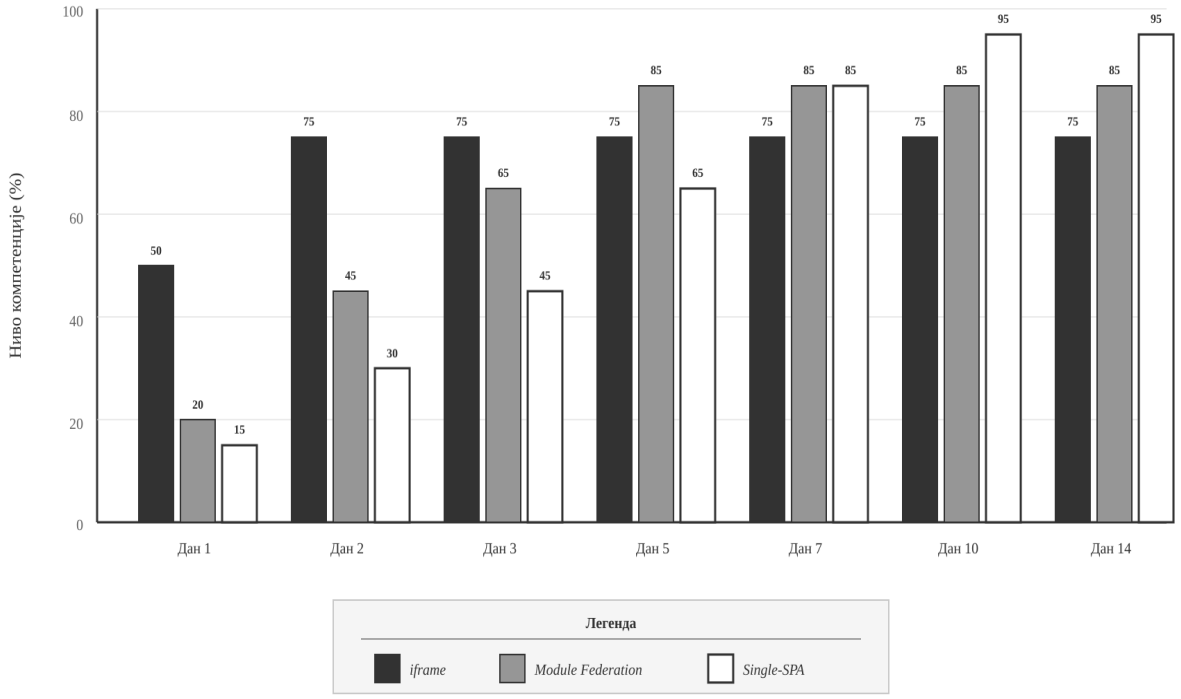
7.3.3. Крива учења

Крива учења је субјективна метрика која зависи од претходног искуства програмера. Процјена је базирана на анкети хипотетичких *React* програмера средње сениорности који немају претходно искуство са микрофронтендима.

Аспект	iframe	Module Federation	Single-SPA
Основни концепти	Лако	Средње	Тешко
Конфигурација	Лако	Средње	Средње
Дебаговање	Тешко	Средње	Средње
Документација	Одлична (MDN)	Добра	Одлична
Community подршка	Ограничена	Растућа	Зрела
Вријеме до продуктивности	1-2 дана	3-5 дана	5-7 дана

Табела 7.7: Процјена криве учења

Iframe приступ има најмању криву учења за основне концепте јер се ослања на стандардне веб технологије (HTML *iframe*, *postMessage* API) које су добро документоване. Међутим, дебаговање је изазовнији због изолације контекста. *Module Federation* захтијева разумијевање *Webpack* специфичних концепата (*chunks*, *module resolution*) али нуди интуитивније дебаговање. *Single-SPA* има највећу почетну криву учења због концепата животног циклуса али има најбољу документацију и подршку заједнице.



Слика 7.4: Дијаграм криве учења током времена за сва три приступа

7.4. Искуство програмера

Developer Experience (DX) је мјера колико је пријатно и продуктивно радити са одређеном технологијом. Добар DX смањује фрустрацију, повећава мотивацију, и убрзава испоруку функционалности.

7.4.1. Процес развоја софтвера

Метрика	*iframe*	*Module Federation*	*Single-SPA*
Покретање сервера	~2.5s (x3)	~3.2s (x4)	~2.8s (x6)
HMR вријеме	~150ms	~200ms	~180ms
Продукциони *build*	~8s	~12s	~15s
Инкрементални *build*	~3s	~4s	~5s
Независни развој	Да	Дјелимично	Да

Табела 7.8: Поређење различитих процеса у развоју софтвера

Iframe приступ омогућава потпуно независан развој гдје сваки микрофронтенд може бити развијан изоловано без потребе за покретањем осталих апликација. *Module Federation* захтијева да *Host* апликација буде покренута за тестирање интеграције *Remote* модула. *Single-SPA* подржава самостални развој уз коришћење *single-spa-inspector* алата за тестирање без *root-config*-а.

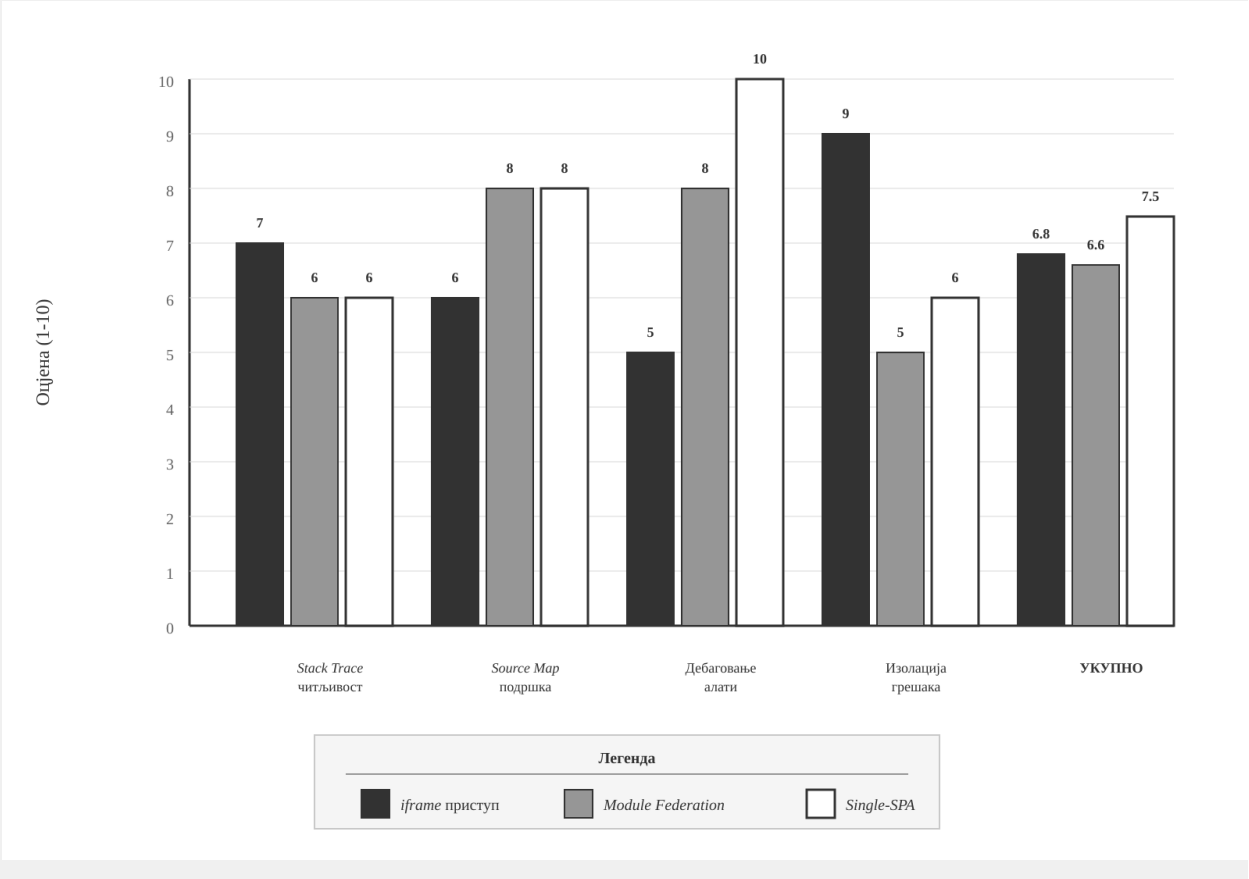
7.4.2. Могућности за дебаговање

Дебаговање у микрофронтенд архитектури је инхерентно сложеније него у монолитним апликацијама због дистрибуиране природе система. Свакі приступ има специфичне изазове и алате за дебаговање.

Iframe дебаговање је најизазовније јер *Chrome DevTools* приказује засебан *JavaScript* контекст за сваки *iframe*. Програмер мора ручно пребацивати између контекста у *Console* и *Sources* панелима. *Network* panel приказује захтјеве свих *iframe*-ова заједно, што може бити збуњујуће. Међутим, изолација значи да грешке у једном *iframe*-у не утичу на дебаговање другог.

Module Federation дебаговање је побољшано коришћењем *source* мапи који омогућавају *mapping minified* кода на оригинални *TypeScript*. Грешке се приказују у јединственом контексту, али траг грешке(*stack trace)* може укључивати динамички учитане модуле што отежава праћење. *React DevTools* функционише нормално и приказује комплетно *component tree*.

Single-SPA дебаговање је олакшао *single-spa:devtools* екстензијом за претраживач која приказује статус свих регистрованих апликација, омогућава ручни *mount*/*unmount*, и логује догађаје у животном циклусом. RxJS *Spy* алат може бити коришћен за праћење *Observable* стања у *Event Bus*-у.



Слика 7.5: Поређење алата за дебаговање за сва три приступа

7.5. Скалабилност

Скалабилност микрофронтенд архитектуре се манифестује у три димензије: скалабилност броја микрофронтенда, скалабилност броја тимова који раде паралелно, и скалабилност у смислу броја корисника који истовремено приступају апликацији.

7.5.1 Скалабилност броја микрофронтенда

Број МФ	*iframe* утицај	MF утицај	*Single-SPA* утицај
3-5	Минималан	Минималан	Минималан
6-10	Умјерен (меморија)	Низак	Низак
11-20	Значајан	Умјерен (*build*)	Умјерен (*layout*)
20+	Критичан	Значајан	Значајан

Табела 7.9: Скалабилност при повећању броја микрофронтенда

Iframe приступ је најмање скалабилан због линеарног раста потрошње меморије и мрежних захтјева са сваким новим *iframe*-ом. При 10+ *iframe*-ова, мобилни уређаји могу доживјети значајно успорење. *Module Federation* и *Single-SPA* се скалирају значајно боље захваљујући дијељењу ресурса, али и они захтијевају пажљиво планирање при 20+ микрофронтенда.

7.5.2. Скалабилност паралелног развоја

Способност вишеструких тимова да раде паралелно без међусобног блокирања је кључна предност микрофронтенд архитектуре. Свакі приступ нуди различит степен независности.

Аспект	*iframe*	*Module Federation*	*Single-SPA*
Независни repo	Могуће	Да	Да
Независни deploy	Да	Да	Да
Технолошка слобода	Потпуна	Ограничена	Потпуна
Координација	Минимална	Средња	Средња
Ризик од грешака	Низак	Висок	Средња

Табела 7.10: Подршка за паралелни развој

Iframe приступ нуди највећу независност тимова захваљујући потпуној изолацији. Тимови могу користити потпуно различите технологије без ризика од конфликата. *Module Federation* захтијева координацију око дијељених зависности и типова, што може бити извор конфликата при честим промјенама. *Single-SPA* је у средини, захтијевајући координацију око *event bus* интерфејса али дозвољавајући технолошку слободу.

7.6. Одржавање и дугорочна одрживост

Дугорочна одрживост архитектуре је критичан фактор који утиче на укупне трошкове власништва (*Total Cost of Ownership*). Анализа укључује лакоћу ажурирања зависности, управљање техничким дуговањем, и изолацију грешака.

7.6.1. Управљање зависностима

Ажурирање зависности у микрофронтенд архитектури може бити комплексније него у монолитној апликацији због потребе за координацијом између модула. Различити приступи нуде различите стратегије за управљање овом сложеношћу.

Iframe приступ омогућава потпуно независно ажурирање зависности. Сваки микрофронтенд може користити произвољну верзију библиотеке без утицаја на остале. Ово је предност за постепену миграцију али може довести до техничког дуговања ако тимови занемаре ажурирања.

Module Federation захтијева координацију при ажурирању дијељених зависности. Ако један *Remote* ажурира *React* на нову верзију, сви остали морају слиједити. *requiredVersion* конфигурација може помоћи у управљању компатибилношћу, али захтијева пажљиво планирање.

Single-SPA са *Import Maps* омогућава централизовано управљање верзијама дијељених библиотека. Промјена URL-а у I*mport Map*-и утиче на све микрофронтенде истовремено, што олакшава координиране надоградње али повећава ризик од грешака.

7.6.2. Изолација грешака

Тип грешке	*iframe*	*Module Federation*	*Single-SPA*
JavaScript runtime грешка	Изолована	Пропагира се	Пропагира се
CSS колизија	Изолована	Могућа	Могућа
Изливање меморије	Изолован	Дијељен	Дијељен
Network грешка	Дјелимична	Дјелимична	Дјелимична
Глобална варијабла	Изолована	Могућ конфликт	Могућ конфликт

Табела 7.11: Изолација грешака по типу

Iframe приступ пружа најбољу изолацију грешака. *JavaScript* грешка у једном *iframe*-у не може срушити остатак апликације. CSS је потпуно изолован, елиминишући ризик од колизија селектора. Изливања меморије су ограничена на појединачни *iframe* контекст.

Module Federation и *Single-SPA* дијеле исто *JavaScript* вријеме извршења, што значи да неухваћена грешка може утицати на цијелу апликацију. *Error Boundary* компоненте могу ублажити овај проблем али не елиминишу га потпуно. CSS захтијева дисциплиноване конвенције именовања или *CSS-in-JS* рјешења за избјегавање колизија.

7.7. Препоруке за избор приступа

На основу спроведене анализе, сљедеће препоруке могу помоћи у избору одговарајућег приступа за специфичне сценарије употребе.

7.7.1. Када изабрати *iframe* приступ

Iframe приступ је препоручен у сљедећим сценаријима: интеграција старијих система који користе застарјеле библиотеке или имају проблематичан CSS; уграђивање компоненти трећих страна (*payment* форме, *chat widget*-и) гдје је сигурносна изолација критична; ситуације гдје различити тимови имају радикално различите технолошке стекове без могућности или жеље за усклађивањем; прототипирање и POC пројекти гдје брзина имплементације има приоритет над перформансама.

Iframe приступ није препоручен за апликације гдје су перформансе и SEO критични, за мобилне апликације са ограниченим ресурсима, нити за апликације са интензивном комуникацијом између компоненти.

7.7.2. Када изабрати *Module Federation*

Module Federation је оптималан избор када сви тимови користе исти радни оквир(посебно *React*); када је оптимизација *bundle* величине приоритет; када је потребна дубока интеграција између микрофронтенда са дијељеним стањем; када постоји централизовани тим за одржавање инфраструктуре који може управљати сложеношћу конфигурације.

Module Federation захтијева опрез у сценаријима са честим промјенама дијељених зависности, када постоји ризик од некоординираних ажурирања између тимова, или када је потребна подршка за радне оквире различите од *React*-а.

7.7.3. Када изабрати *Single-SPA*

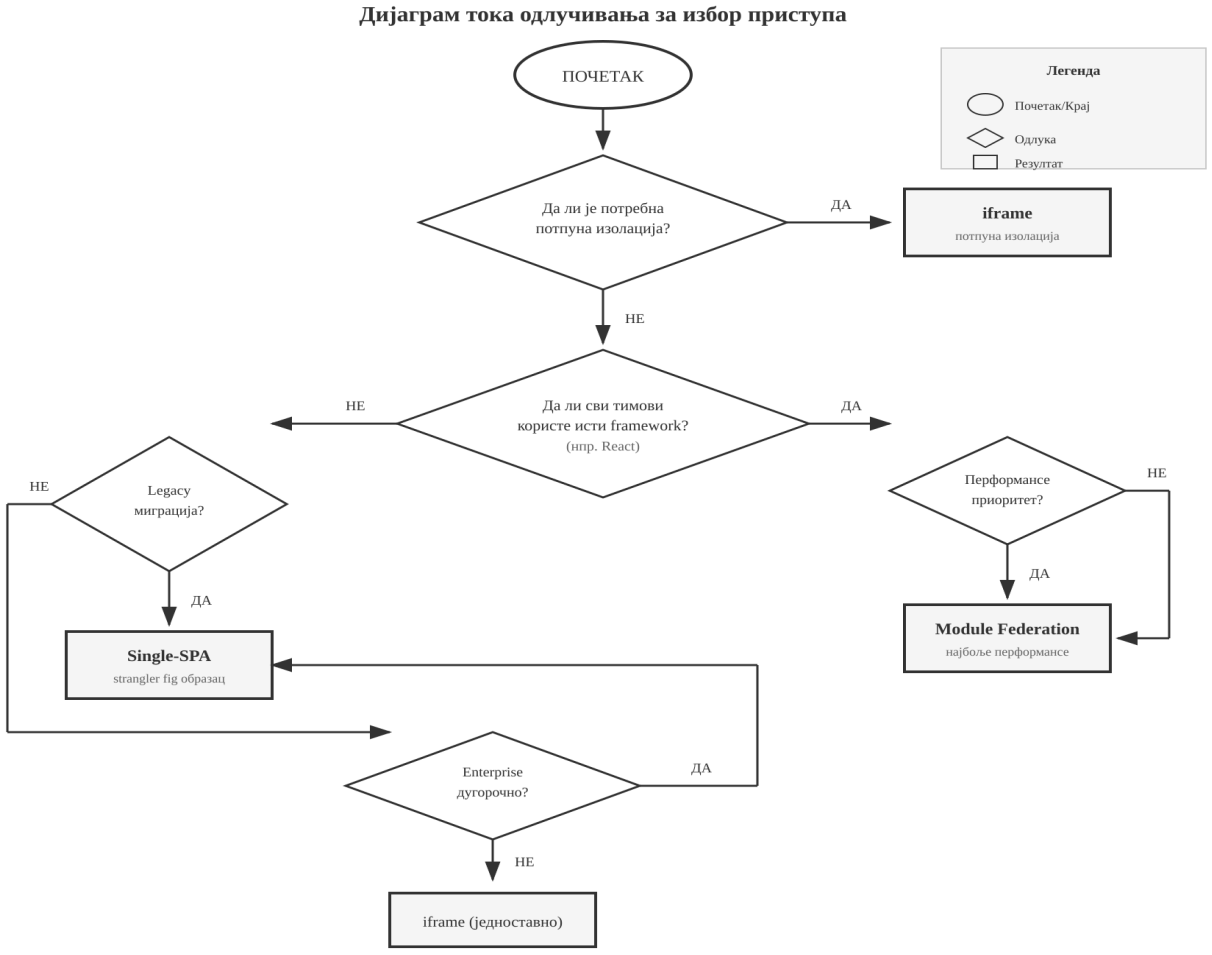
Single-SPA је најбољи избор за постепену миграцију старијих апликација ("*strangler fig*" образац); када је потребна подршка за вишеструке радне оквире унутар исте апликације; када тимови захтијевају баланс између независности и интеграције; када је важна зрелост екосистема и подршка заједнице.

Single-SPA захтијева већу почетну инвестицију у учење и конфигурацију, што га чини мање погодним за мале пројекте или тимове без искуства са микрофронтендима.

7.7.4. Матрица одлучивања

Сценарио	Препоручени приступ	Образложење
Legacy интеграција	*iframe*	Потпуна изолација спречава конфликте
Један радни оквир	*Module Federation*	Најбоље перформансе, дијељење контекста
Хетерогени радни оквир	*Single-SPA*	Агностицизам радног оквира, зрео екосистем
Максималне перформансе	*Module Federation*	Најефикасније дијељење кода
Максимална изолација	*iframe*	Потпуна *sandbox* изолација
Постепена миграција	*Single-SPA*	*Strangler fig* образац подршка
Мали тим, брзо	*iframe*	Најмања крива учења
Велики пројекти, дугорочно	*Single-SPA*	Зрелост, подршка, скалабилност

Табела 7.12: Матрица одлучивања за избор приступа



Слика 7.6: Дијаграм тока одлучивања за избор приступа

7.8. Закључак компаративне анализе

Компаративна анализа три приступа микрофронтенд архитектури показује да не постоји универзално најбоље рјешење. Сваки приступ има своје предности и недостатке који га чине оптималним за одређене сценарије.

Iframe приступ, иако најстарији и често сматран застарјелим, остаје вриједан избор за сценарије гдје је изолација примарни захтјев. Његова способност да потпуно изолује CSS, *JavaScript*, и DOM чини га незамјењивим за интеграцију непоузданих или старијих компоненти.

Module Federation представља модеран приступ који нуди најбоље перформансе захваљујући ефикасним дијељењу коа у вријеме извршења. Његова дубока интеграција са *React* екосистемом чини га идеалним за тимове који су стандардизовани на *React*-у и желе максимизовати ефикасност.

Single-SPA пружа најфлексибилније рјешење са зрелим екосистемом и одличном документацијом. Његов агностицизам за радне оквире и софистицирано управљање животним циклусом чине га погодним за комплексне и велике апликације са дугорочном визијом. Коначан избор треба бити вођен специфичним захтјевима пројекта, постојећом технолошком инфраструктуром, вјештинама тима, и дугорочном визијом производа. Важно је напоменути да ови приступи нису међусобно искључиви - хибридне архитектуре које комбинују више приступа могу бити оптимално рјешење за комплексне системе.

8. Закључак

Микрофронтенд архитектура представља значајан помак у начину на који се развијају и одржавају сложене веб апликације. Примјеном принципа микросервисне архитектуре на фронтенд слој, омогућено је да велики тимови раде паралелно на истој апликацији уз минималну међусобну зависност, да се технолошки стек постепено модернизује без потребе за потпуним преписивањем, и да се смањи ризик при испоруци нових функционалности кроз независне циклусе испоруке.

Овај дипломски рад је систематски истражио три доминантна приступа имплементацији микрофронтенд архитектуре: интеграцију путем *iframe* елемената, *Webpack Module Federation*, и *Single-SPA* мета радни оквир. Кроз теоријску анализу, практичну имплементацију идентичне е-продавница апликације (*B-World*), и детаљну компаративну евалуацију, стечен је дубок увид у карактеристике, предности и ограничења сваког приступа.

8.1. Преглед остварених циљева

Примарни циљ рада, упоредна анализа три приступа микрофронтенд архитектури, је успјешно остварен. Анализа је обухватила теоријске основе сваког приступа, практичну имплементацију функционалне апликације, и мјерење перформанси у контролисаним условима. Резултати анализе пружају конкретне смјернице за избор приступа у реалним пројектима.

Секундарни циљеви рада су такође испуњени. Демонстрирана је практична примјена микрофронтенд образаца кроз три независне имплементације. Идентификоване су специфичне ситуације у којима је сваки приступ оптималан. Документовани су кључни изазови и најбоље праксе за сваки приступ.

Циљ	Статус	Резултат
Теоријска анализа приступа	Остварен	Поглавље 3 - детаљан преглед механизама
Практична имплементација	Остварен	Три функционалне апликације на *GitHub*-у
Мјерење перформанси	Остварен	*Lighthouse* и *Chrome DevTools* анализа
Компаративна евалуација	Остварен	12 табела упоредних метрика
Препоруке за избор	Остварен	Матрица одлучивања за 8 сценарија

Табела 8.1: Преглед остварених циљева истраживања

8.2. Кључни налази истраживања

Истраживање је потврдило да не постоји универзално најбољи приступ микрофронтенд архитектури. Сваки од три анализирана приступа има јасно дефинисану нишу у којој је оптималан избор.

8.2.1. Налази о *iframe* приступу

Iframe приступ је потврђен као најједноставнији за имплементацију и једини који пружа потпуну изолацију између микрофронтенда. Ова изолација се показала као неопходна у сценаријима интеграције старијих система или компоненти трећих страна. Међутим, цијена ове изолације је значајна: укупна величина *bundle*-а је била готово три пута већа

(*949 KB* наспрам *334 KB* за *Module Federation*), а *Lighthouse Performance* оцјена је била најнижа (62 наспрам 89).

Посебно је интересантан налаз да *iframe* приступ омогућава коришћење потпуно различитих радних оквира (*React, Svelte, Vue*) без икаквих компромиса. Ово је демонстрирано у имплементацији гдје је *Container* био у *React*-у, *Book List* у *Svelte*-у, а *Single Book* у *Vue*-у. Ниједан други приступ није понудио овај ниво технолошке слободе без додатне сложености.

8.2.2. Налази о *Module Federation* приступу

Module Federation се показао као најефикаснији приступ у погледу перформанси. Дијељење *React*-а у вријеме извршавања и других библиотека резултирало је најмањом укупном величином *bundle*-а и најбољим *Lighthouse* оцјенама у свим категоријама. Дијељени *React* контекст је омогућио елегантну комуникацију између микрофронтенда без потребе за експлицитним размјенама порука.

Кључно ограничење *Module Federation* приступа је везаност за исти радни оквир. У имплементацији су сви модули морали користити *React*, што ограничава флексибилност при интеграцији постојећих апликација у различитим технологијама. Такође је примијећено да конфигурација дијељених зависности захтијева пажљиву координацију између тимова како би се избјегли конфликти верзија.

8.2.3. Налази о *Single-SPA* приступу

Single-SPA је потврђен као најзрелије и најфлексибилније рјешење за сценарије великих апликација. Његов агностицизам за радни оквир, софистицирано управљање животним циклусом, и богат екосистем помоћних библиотека чине га идеалним за дугорочне пројекте са комплексним захтјевима. RxJS *Event Bus* образац за комуникацију се показао као елегантан и скалабилан.

Главни изазов *Single-SPA* приступа је почетна крива учења. Концепти попут функција животног циклуса (bootstrap, mount, unmount), *Import Maps*, и *layout engine*-а захтијевају вријеме за савладавање. Процијењено вријеме до продуктивности од 5-7 дана је значајно дуже од *iframe* приступа (1-2 дана), што може бити проблематично за пројекте са ограниченим временским оквиром.

8.3. Доприноси рада

Овај дипломски рад пружа неколико конкретних доприноса области микрофронтенд архитектуре:

Прво, рад нуди систематску класификацију и упоређење три доминантна приступа на основу јединствене методологије. За разлику од постојеће литературе која се типично фокусира на појединачне приступе, овај рад омогућава директну упоредбу коришћењем идентичне апликације као основе за евалуацију.

Друго, развијене су три комплетне, функционалне имплементације које су јавно доступне на *GitHub*-у. Ове имплементације могу служити као референтне архитектуре за програмере који разматрају усвајање микрофронтенд приступа. Код је документован и организован према најбољим праксама индустрије.

Треће, креирана је матрица одлучивања која мапира специфичне сценарије употребе на препоручене приступе. Ова матрица представља практичан алат за архитекте и техничке лидере који треба да донесу одлуку о избору приступа.

Четврто, документовани су конкретни обрасци комуникације за сваки приступ: *postMessage* за *iframe*, *React Context* за *Module Federation*, и RxJS *BehaviorSubject* за *Single-SPA*. Ови обрасци могу бити директно примијењени у реалним пројектима.

8.4. Ограничења истраживања

Истраживање има неколико ограничења која треба узети у обзир при интерпретацији резултата:

Тестирање перформанси је спроведено на апликацији умјерене сложености са три микрофронтенда. Резултати се могу значајно разликовати за апликације са 10+, 20+ или више микрофронтенда гдје ефекти скалабилности постају израженији.

Сва мјерења су вршена у развојном и симулираном продукционом окружењу на локалној машини. Реално продукцијско окружење са CDN дистрибуцијом, географски дистрибуираним корисницима, и варијацијама мрежних услова може показати другачије резултате.

Евалуација је фокусирана на техничке аспекте. Организациони фактори попут структуре тимова, постојеће инфраструктуре, и корпоративне културе такође значајно утичу на успјех микрофронтенд иницијативе али нису били предмет овог истраживања.

Iframe имплементација је користила различите радне оквире (*React*, *Svelte*, *Vue*) док су *Module Federation* и *Single-SPA* користили само *React*. Ово је намјерна одлука која одражава типичну употребу сваког приступа, али онемогућава потпуно изоловану компарацију агностицизма према радним оквирима.

8.5. Препоруке за будућа истраживања

На основу искуства стеченог током овог истраживања, идентификовано је неколико смјерова за будући рад:

Истраживање хибридних приступа који комбинују елементе различитих техника. На примјер, коришћење *iframe*-а за застарјеле компоненте унутар *Single-SPA* оркестриране апликације, или комбиновање *Module Federation*-а са *Import Maps* за флексибилније дијељење зависности.

Анализа утицаја микрофронтенд архитектуре на *Server-Side Rendering* (SSR) и *Static Site Generation* (SSG). Ови обрасци постају све важнији за перформансе и SEO, а њихова интеграција са микрофронтендима представља отворено истраживачко питање.

Развој аутоматизованих алата за мониторинг и оптимизацију микрофронтенд апликација. Постојећи алати су углавном дизајнирани за монолитне апликације и не адресирају специфичне изазове дистрибуиране фронтенд архитектуре.

Истраживање сигурносних импликација микрофронтенд архитектуре, посебно у контексту дијељеног вријемена извршавања (*Module Federation*, *Single-SPA*) гдје злонамјерни или компромитовани микрофронтенд може утицати на цијелу апликацију.

Лонгитудинална студија одржавања микрофронтенд апликација током вишегодишњег периода. Теоријске предности у погледу независних ажурирања и смањеног техничког дуговања захтијевају емпиријску валидацију у реалним продукционим окружењима.

8.6 Закључак

Микрофронтенд архитектура није сребрни метак који рјешава све проблеме фронтенд развоја. Она уводи нову класу сложености у замјену за предности скалабилности и независности тимова. Одлука о усвајању микрофронтенд приступа треба бити пажљиво размотрена, узимајући у обзир величину и сложеност апликације, структуру и величину тима, постојећу техничку инфраструктуру, и дугорочне планове за развој производа.

За организације које се суочавају са изазовима координације великих фронтенд тимова, постепене миграције застарјелих система, или потребе за независним циклусима испоруке, микрофронтенд архитектура нуди провјерен приступ са растућим екосистемом алата и заједнице. Избор конкретне технике - *iframe*, *Module Federation*, или *Single-SPA* - треба бити вођен специфичним контекстом пројекта, а не трендовима индустрије.

Овај рад је показао да сва три приступа имају своје мјесто у савременом фронтенд развоју. *Iframe* остаје релевантан за сценарије максималне изолације. *Module Federation* нуди оптималне перформансе за *React* екосистем. *Single-SPA* пружа највећу флексибилност за хетерогене и дугорочне пројекте. Разумијевање предности и ограничења сваког приступа омогућава информисану одлуку која ће допринијети успјеху пројекта.

На крају, важно је напоменути да је микрофронтенд архитектура област у активном развоју. Нове технике и алати континуирано настају, а постојећи се унапређују. *Module Federation 2.0 (Rspack)*, *Native Federation* за *Angular*, и напредак у стандардима веб компоненти само су неки од развоја који обећавају још боље могућности у будућности. Програмери и архитекте који раде у овој области требају остати информисани о новим развојима и бити спремни прилагодити своје приступе како технологија напредује.

Правци даљег развоја укључују примјену вјештачке интелигенције за самостални избор и подешавање микрофронтенд приступа на основу потреба пројекта. Планирано је коришћење *machine learning* модела за унапријед откривање проблема који настају међусобним утицајем независних модула. Такође, испитаће се могућности AI-вођеног прилагођавања корисничког искуства кроз усмјерено учитавање компоненти према потребама сваког корисника.

Референце

[1] Newman, S. (2021). *Building Microservices: Designing Fine-Grained Systems* (2nd ed.). *O'Reilly Media*. ISBN: 9781492034025. *https://www.oreilly.com/library/view/building-microservices-2nd/9781492034018/*

[2] Geers, M. (2020). *Micro Frontends in Action*. *Manning Publications*. ISBN: 9781617296871. *https://www.manning.com/books/micro-frontends-in-action*

[3] Rappl, F. (2021). *The Art of Micro Frontends*. *Packt Publishing*. ISBN: 9781800563568. *https://www.packtpub.com/product/the-art-of-micro-frontends/9781800563568*

[4] Mezzalira, L. (2021). *Building Micro-Frontends*. *O'Reilly Media*. ISBN: 9781492082996. *https://www.oreilly.com/library/view/building-micro-frontends/9781492082989/*

[5] *Meta Platforms, Inc.* (2024). *React Documentation*. *https://react.dev/,* Датум последњег приступа: 10.03.2026

[6] *Evan You & Vue.js Contributors.* (2024). *Vue.js Documentation*. *https://vuejs.org/guide/introduction.html,* Датум последњег приступа: 10.03.2026

[7] *Svelte Contributors.* (2024). *Svelte Documentation*. *https://svelte.dev/docs,* Датум последњег приступа: 10.03.2026

[8] *Single-SPA Contributors.* (2024). *Single-SPA Documentation*. *https://single-spa.js.org/docs/getting-started-overview,* Датум последњег приступа: 10.03.2026

[9] *Webpack Contributors.* (2024). *Module Federation Documentation*. *https://webpack.js.org/concepts/module-federation/,* Датум последњег приступа: 10.03.2026

[10] *Evan You & Vite Contributors.* (2024). *Vite Documentation*. *https://vitejs.dev/guide/* Датум последњег приступа: 10.03.2026

[11] *RxJS Contributors.* (2024). *RxJS Documentation*. *https://rxjs.dev/guide/overview,* Датум последњег приступа: 10.03.2026

[12] *MDN Web Docs.* (2024). *Window: postMessage() method*. *Mozilla Developer Network*. *https://developer.mozilla.org/en-US/docs/Web/API/Window/postMessage,* Датум последњег приступа: 10.03.2026

[13] *MDN Web Docs.* (2024). *<iframe>*: *The Inline Frame element*. *Mozilla Developer Network*. *https://developer.mozilla.org/en-US/docs/Web/HTML/Element/iframe,* Датум последњег приступа: 10.03.2026

[14] Jackson, C. (2019). *Micro Frontends*. *martinfowler.com*. *https://martinfowler.com/articles/micro-frontends.html,* Датум последњег приступа: 10.03.2026

[15] Geers, M. (2017). *Micro Frontends - extending the microservice idea to frontend development*. *micro-frontends.org*. *https://micro-frontends.org/,* Датум последњег приступа: 10.03.2026

[16] *ThoughtWorks.* (2016). *Technology Radar - Micro Frontends*. *ThoughtWorks Technology Radar*. *https://www.thoughtworks.com/radar/techniques/micro-frontends,* Датум последњег приступа: 10.03.2026

[17] Pavlenko, A. (2020). *Webpack 5 Module Federation: A game-changer in JavaScript architecture*. *Medium*. *https://medium.com/swlh/webpack-5-module-federation-a-game-changer-to-javascript-architecture-bcdd30e02669,* Датум последњег приступа: 10.03.2026

[18] Jackson, Z. (2020). *Module Federation Documentation*. *module-federation.github.io*. *https://module-federation.github.io/,* Датум последњег приступа: 10.03.2026

[19] Denning, J. (2019). *A step-by-step guide to single-spa*. *single-spa.js.org Blog*. *https://single-spa.js.org/docs/recommended-setup/,* Датум последњег приступа: 10.03.202

Биографија



Младен Бајић рођен је 1999. године у Бијељини, Босна и Херцеговина. Основну школу Кнез Иво од Семберије завршио је 2015. године у Бијељини. Средњу економску школу, смјер пословно-правни техничар, завршио је 2018. године у Бијељини.

Школске 2018/2019. године уписао је Факултет техничких наука, Универзитет у Новом Саду, смер Инжењерство информационих система. Током студија постигао је просјечну оцјену 8.65 и показао посебно интересовање за области веб технологија, микрофронтенд архитектуре, *frontend* развоја и савремених *JavaScript* радних оквира.

Током студија стекао је вишегодишње радно искуство у софтверској индустрији са фокусом на *frontend* развој. Радио је као софтверски инжењер у компанији ServalIT (2021-2025), веб програмер у компанији RTRK (2025), те софтверски инжењер у компанији Avantos.ai. Током рада остварио се у позицијама вође тима и ментора.

Поседује напредно знање енглеског језика.