|  |  |  |
| --- | --- | --- |
|  | УНИВЕРЗИТЕТ  У НОВОМ САДУ  **ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА У НОВОМ САДУ** |  |

Владислав Петковић

**Микросервисна архитектура - QuizHub**

ПРОЈЕКАТ

Основне академске студије

Нови Сад, 2025

**Садржај**

[1. Увод 1](#_Toc210164939)

[2. Опис коришћених технологија и алата 2](#_Toc210164940)

[2.1. Frontend технологије 2](#_Toc210164941)

[2.2. Технологије задње стране (Backend) 2](#_Toc210164942)

[2.3. База података и ORM 3](#_Toc210164943)

[*Конфигурација контекста базе података за табели корисника* 4](#_Toc210164944)

[2.4. Контејнеризација и DevOps 4](#_Toc210164945)

[2.5. API Gateway - Ocelot 5](#_Toc210164946)

[*Пример конфигурације улаза и излаза Gateway-а* 5](#_Toc210164947)

[3. Опис решења 6](#_Toc210164948)

[3.1. Архитектура система 6](#_Toc210164949)

[3.2. Имплементација микросервиса и архитектурни шаблони 6](#_Toc210164950)

[3.2.1. User Service - SOLID принципи 6](#_Toc210164951)

[3.2.2. Quiz Service - Domain-Driven Design 7](#_Toc210164952)

[3.3. Service-to-Service комуникација 8](#_Toc210164953)

[3.3.1. HTTP Client имплементација 8](#_Toc210164954)

[3.3.2. Шаблони резилијенције са Polly 8](#_Toc210164955)

[3.3.3. Откривање сервиса 8](#_Toc210164956)

[3.3.4. Руковање грешкама у међусервисној комуникацији 8](#_Toc210164957)

[3.3.5. Оптимизација перформанси 9](#_Toc210164958)

[3.4. API Gateway - детаљна имплементација 10](#_Toc210164959)

[3.4.1. Request Routing 10](#_Toc210164960)

[3.4.2. JWT Authentication на Gateway нивоу 10](#_Toc210164961)

[3.4.3. CORS политика 10](#_Toc210164962)

[3.4.4. Request/Response Logging 10](#_Toc210164963)

[3.4.5. Load Balancing саобраћаја 11](#_Toc210164964)

[3.5. Безбедност и аутентикација 12](#_Toc210164965)

[3.5.1. Хеширање лозинки са HMACSHA512 12](#_Toc210164966)

[3.5.2. JWT Token аутентикација 12](#_Toc210164967)

[3.5.3. Role-Based Access Control - Приступ на основу улога 12](#_Toc210164968)

[3.5.4. Валидација и чишћење уноса и параметара 12](#_Toc210164969)

[3.5.5. Заштита од честих претњи 13](#_Toc210164970)

[3.6. Deployment и скалабилност 14](#_Toc210164971)

[3.6.1. Умрежавање контејнера 14](#_Toc210164972)

[3.6.2. Перзистенција података кроз Volume-e 14](#_Toc210164973)

[3.6.3. Мониторинг здравља сервиса 14](#_Toc210164974)

[3.6.4. Хоризонтално скалирање 14](#_Toc210164975)

[4. Закључак 15](#_Toc210164976)

[Literatura 16](#_Toc210164977)

# Увод

Овај документ представља техничку документацију за **QuizHub** апликацију - модерну веб платформу за креирање, управљање и извршавање квизова, развијену коришћењем микросервисне архитектуре. У данашњем свету где се образовање све више помера ка дигиталним платформама, постоји растућа потреба за ефикасне и скалабилне системе који могу да подрже велики број корисника који истовремено приступају образовним садржајима.

**QuizHub** платформа решава проблем креирања централизованог система за управљање квизовима који може да подржи различите типове корисника (студенте, наставнике, администраторе) са различитим нивоима приступа и функционалности. Традиционални монолитни приступи развоја оваквих система често се суочавају са проблемима скалабилности, одржавања и независног развоја различитих функционалности.

Главни циљ овог пројекта је био развој модуларне, скалабилне архитектуре која омогућава:

* Независан развој и *deployment* различитих делова система
* Ефикасно управљање ресурсима кроз контејнеризацију
* Висок ниво безбедности корисничких података
* Интуитиван кориснички интерфејс развијен модерним *frontend* технологијама

Микросервисна архитектура је изабрана као основни архитектурни образац због њених предности у контексту образовних платформи где различити делови система (управљање корисницима, садржај квизова, резултати) имају различите захтеве за перформансе и скалабилност.

**Предметни пројекат** представља проширење основног пројекта из предмета *Примена веб програмирања у инфраструктурним системима*. Основна апликација је надограђена микросервисном архитектуром, где је монолитни систем подељен на четири независна сервиса: User Service, Quiz Service, Result Service и API Gateway. Овај приступ омогућава независно скалирање сваког дела система према потребама оптерећења.

# Опис коришћених технологија и алата

У оквиру овог поглавља биће детаљно описане све технологије и алати коришћени у развоју **QuizHub** платформе, са образложењем избора и њиховим кључним карактеристикама.

## Frontend технологије

За развој корисничког интерфејса изабран је **React** *framework* у верзији 19.1.1, који представља једну од најпопуларнијих и најстабилнијих технологија за развој модерних веб апликација [1]. **React** је изабран због следећих кључних предности:

**TypeScript интеграција** - Коришћена је верзија 4.9.5 која обезбеђује статичко типизирање и боље искуство током писања кода (*developer experience*). **TypeScript** значајно смањује број грешака у току развоја и олакшава одржавање кода [2]. Интерфејси и типови су дефинисани за све ентитете система, укључујући User, Quiz, Question, Answer, Result и DTO моделе.

**SCSS модули** - За стилизовање је коришћен **SCSS** (*Syntactically Awesome Style Sheets*) који омогућава модуларан приступ дефинисању стилова и боље организовање CSS кода [3]. Свака компонента има свој SCSS модул који се аутоматски скопује и не конфликтује са другим стиловима.

**React Router DOM** (верзија 7.7.1) - Обезбеђује клијентско рутирање што омогућава *Single Page Application* (SPA) функционалност без освежавања целе странице [4]. Имплементиране су заштићене руте које проверавају аутентификацију и ауторизацију пре приказивања компоненти.

**Recharts библиотека** (верзија 3.1.2) - Специјализована за креирање интерактивних графикона и визуелизација статистичких података о перформансама корисника [5]. Користи се за приказ историје резултата и анализу перформанси на квизовима.

## Технологије задње стране (*Backend*)

За развој задње стране изабран је **.NET 8.0** као основни радни оквир (*framework*) за развој микросервиса. Овај избор је мотивисан следећим факторима:

**Перформансе и скалабилност** - **.NET 8.0** представља најновију *Long Term Support* (LTS) верзију која обезбеђује високе перформансе и оптимизовано извршавање [6]. Подршка до новембра 2026. године гарантује стабилност и безбедносне ажурирање.

**Микросервисна подршка** - Уграђена подршка за развој микросервиса кроз минималне API-е (*Application Programming Interface*), инјектовање зависности (*dependency injection - DI*), и конфигурацију [7]. Сваки сервис користи *DI* за управљање зависностима и олакшано тестирање.

***Cross-platform* подршка** - Могућност извршавања на различитим оперативним системима кроз **.NET Core** *runtime* [8]. Сервиси се могу извршавати на Windows, Linux и macOS платформама без промене кода.

Архитектура сваког микросервиса следи принципе **чисте архитектуре** (*Clean Architecture*) са јасно дефинисаним слојевима: Домен (ентитети), Апликација (сервиси), и Презентација (API контролери). Овај приступ омогућава лако тестирање и одржавање кода.

public class User

{

public Guid Id { get; set; }

[Required] public required string Username { get; set; }

[Required] public required string Email { get; set; }

[Required] public required byte[] PasswordHash { get; set; }

[Required] public required byte[] PasswordSalt { get; set; }

public byte[] AvatarImage { get; set; }

public required string Role { get; set; }

}

## База података и ORM

**SQL Server 2022** је изабран као главни систем за управљање базом података због своје робусности, скалабилности и одличне интеграције са **.NET** екосистемом [9]. Коришћена је *Express* верзија која је бесплатна и довољна за образовне и развојне сврхе.

***Entity Framework Core*** (*EF Core*) у верзији 9.0.7 служи као *Object-Relational Mapping* (ORM) алат који омогућава:

* Аутоматско генерисање SQL (*Structured Query Language*) упита из C# кода са оптимизацијом перформанси
* *Code-first* приступ где C# класе дефинишу структуру базе података
* Аутоматске миграције база података са верзионисањем и праћењем промена [10]
* LINQ подршку за изражајне и типизоване упите

Имплементиран је ***Database-per-Service* (база по сервису)** образац где сваки микросервис има независну базу података. **UserService** користи *UserDb*, **QuizService** користи *QuizDb*, а **ResultService** користи *ResultDb*. Овај приступ обезбеђује потпуну изолацију података и омогућава независно скалирање база података.



Конфигурација контекста базе података за табели корисника

## Контејнеризација и DevOps

**Docker** технологија је коришћена за контејнеризацију свих компоненти система, што обезбеђује конзистентно окружење за развој, тестирање и продукцију [11]. Сваки сервис и свака база података раде у одвојеним контејнерима са дефинисаним ресурсним ограничењима.

**Docker Compose** омогућава оркестрацију више контејнера и дефинисање међузависности сервиса. Једном командом могуће је покренути целокупан систем са свим сервисима и базама података. Docker мрежа обезбеђује комуникацију између контејнера док су они изоловани од спољашњег света осим кроз дефинисане портове.

## API Gateway - Ocelot

***Ocelot API Gateway*** је изабран као решење за API Gateway() функционалност због своје лаке конфигурације и одличне интеграције са **.NET** екосистемом [12]. Ocelot представља *lightweight* (незахтевни) .NET API Gateway специјално дизајниран за микросервисне архитектуре.

**Кључне карактеристике Ocelot-а:**

* **Request Routing** - Рутирање захтева на основу URL шаблона са подршком за параметре и *wildcard-е*
* **Load Balancing** - Дистрибуција оптерећења између вишеструких инстанци истог сервиса
* **Authentication** - Централизована аутентикација са JWT(*Json Web Token*) токенима
* **Rate Limiting** - Ограничавање броја захтева по кориснику



Пример конфигурације улаза и излаза Gateway-а

# Опис решења

## Архитектура система

**QuizHub** систем је заснован на микросервисној архитектури која се састоји од три главна сервиса и једног API Gateway-а. Систем је дизајниран према принципима *Domain-Driven Design* (DDD) и *Clean Architecture* шаблонима.

Архитектура обезбеђује:

* **Слабо спрегнуте сервисе** - сваки сервис може бити развијан и deployован независно без утицаја на друге сервисе
* **Хоризонталну скалабилност** - могућност додавања нових инстанци сервиса на основу оптерећења
* **Fault tolerance** - отказ једног сервиса не утиче на рад осталих делова система захваљујући изолацији
* **Technology heterogeneity** - могућност коришћења различитих технологија за различите сервисе

## Имплементација микросервиса и архитектурни шаблони

### User Service - SOLID принципи

**User Service** је одговоран за управљање корисничким профилима, аутентикацију и ауторизацију. Сервис је имплементиран према **SOLID принципима** који обезбеђују одржавање и проширивање кода:

**Single Responsibility Principle (SRP)** - Свака класа има јасно дефинисану одговорност. *IUserService* се бави CRUD операцијама, *IAuthService* се бави аутентикацијом и регистрацијом, а *IUserValidationService* валидира корисничке податке.

**Open/Closed Principle** - Класе су отворене за проширивање али затворене за модификацију. Нове функционалности се додају кроз нове имплементације интерфејса.

**Liskov Substitution Principle** - Све имплементације интерфејса могу се заменити без утицаја на исправност програма.

**Interface Segregation Principle** - Клијенти не зависе од интерфејса које не користе. Интерфејси су мали и фокусирани.

**Dependency Inversion Principle** - Зависности су усмерене ка апстракцијама (интерфејси) а не ка конкретним имплементацијама. *Dependency Injection* контејнер управља животним циклусом објеката.

**Repository Pattern** кроз *DbContext* апстрахује приступ подацима и омогућава лакшу замену извора података. Све операције над базом података иду кроз *UserDbContext* који имплементира *IUserDbContext* интерфејс.

### Quiz Service - Domain-Driven Design

**Quiz Service** имплементира **Domain-Driven Design** принципе са богатим доменским моделом. Ентитети *Quiz*, *Question* и *Answer* садрже не само податке већ и пословну логику.

**Aggregate Root** - Quiz ентитет је агрегат који контролише приступ *Question* и *Answer* ентитетима. Све промене на питањима и одговорима иду кроз *Quiz* ентитет.

**Value Objects** - *Difficulty* enum представља value object који енкапсулира концепт тежине квиза.

**Service Layer Pattern** - Пословна логика је енкапсулирана у *IQuizService* интерфејсу и његовој имплементацији.

## Service-to-Service комуникација

Међусервисна комуникација у **QuizHub** систему је имплементирана кроз HTTP REST API позиве. Result Service, као једини сервис који комуницира са другим сервисима, користи специјализоване клијенте за комуникацију са User и Quiz сервисима.

### HTTP Client имплементација

Result Service користи **HttpClient** за синхрону комуникацију са другим сервисима. Клијенти су имплементирани као типизирани HTTP клијенти са *dependency injection* подршком:

**IUserServiceClient** - Преузима информације о корисницима (име, презиме) за приказ на ранг листи. Позива *User Service* преко */api/users/{userId}.*

**IQuizServiceClient** - Преузима информације о квизовима и питањима за валидацију одговора. Позива *Quiz Service* на endpoint */api/quizzes/{quizId}.*

### Шаблони резилијенције са Polly

За повећање поузданости међусервисне комуникације, имплементирани су *resilience patterns* користећи **Polly** библиотеку (верзија 8.3.1):

* **Retry Policy** - Аутоматско поновно покушавање неуспелих захтева (до 3 пута) са стратегијом експоненцијалног чекања
* **Circuit Breaker** - Заштита од преоптерећења недоступних сервиса. После 5 узастопних неуспеха, прекида позиве на 30 секунди
* **Timeout Policy** - Ограничава време чекања на одговор (10 секунди по захтеву)

### Откривање сервиса

У *Docker* окружењу, сервиси користе **DNS-базирано откривање сервиса**. *Docker Compose* аутоматски креира DNS записе за сваки сервис, омогућавајући комуникацију коришћењем **имена сервиса** (*user-service, quiz-service*) уместо IP адреса. Ово олакшава скалирање и замену инстанци сервиса.

### Руковање грешкама у међусервисној комуникацији

Имплементиран је стандардизован приступ руковању грешкама:

* **Graceful Degradation** - Ако сервис за кориснике није доступан, сервис за резултате ће сачувати резултат али ранг листа неће бити комплетна
* **Fallback Responses** - У случају недоступности сервиса, враћају се подразумеване предефинисане вредности
* **Centralized Logging** - Све грешке у комуникацији се бележе са детаљима о захтеву и одговору

### Оптимизација перформанси

За оптимизацију перформанси међусервисне комуникације:

* **Connection Pooling** - *HttpClient* користи *connection pooling* за поновну употребу TCP конекција
* **Async/Await** - Сви HTTP позиви су асинхрони да не блокирају *thread pool*
* **Selective Data Transfer** - Параметар *includeImage=false* спречава трансфер профилних слика када нису потребне

## API Gateway - детаљна имплементација

***Ocelot API Gateway*** служи као јединствена тачка приступа за све клијентске захтеве. *Gateway* имплементира неколико кључних функционалности:

### Request Routing

*Gateway* рутира захтеве на основу URL путање. Рутирање је конфигурисано кроз ***ocelot.json*** фајл који дефинише мапирање између *upstream* (клијент) и *downstream* (сервис) путања:

* */api/users/\** се рутира на *user-service:80/api/users/\**
* */api/quizzes/\** се рутира на *quiz-service:80/api/quizzes/\**
* */api/results/\** се рутира на *results-service:80/api/results/\**

### JWT Authentication на Gateway нивоу

*Gateway* имплементира **централизовану** **JWT** **аутентикацију**. Сви захтеви (осим пријаве и регистрације) морају садржати валидан JWT токен у ***Authorization*** header-у. *Gateway* валидира токен пре прослеђивања захтева микросервисима:

* **Token Validation** - Провера потписа токена, валидности и истека
* **Claims Extraction** - Извлачење идентификатора корисника и улоге из токена
* **Authorization Check** - Провера да ли корисник има дозволу за тражени ресурс

### CORS политика

*Gateway* дефинише CORS политику (Cross Origin Resource Sharing – дељење ресурса из различитих извора) која дозвољава захтеве са предње стране апликације. У развојном окружењу, дозвољени су захтеви са *localhost:3000*, док у продукцијском окружењу захтеви долазе са *frontend:80* контејнера.

### Request/Response Logging

*Gateway* бележи све захтеве са следећим информацијама: *Request ID* (*GUID*), HTTP метода, путања, IP адреса клијента, статус кôд одговора, време обраде захтева. Ово омогућава детаљно праћење свих захтева кроз систем.

### Load Balancing саобраћаја

Иако тренутно постоји само једна инстанца сваког сервиса, *Ocelot* подржава *load balancing* између вишеструких инстанци. Када се додају нове инстанце, *Gateway* аутоматски дистрибуира оптерећење користећи *Round Robin* алгоритам.

## Безбедност и аутентикација

Систем имплементира вишеслојни приступ безбедности са фокусом на заштиту корисничких података и превенцију неауторизованог приступа.

### Хеширање лозинки са HMACSHA512

Лозинке се никада не чувају у *plain text* (чист текст) формату. Приликом регистрације, лозинка се **хешира** користећи HMACSHA512 алгоритам са криптографски сигурним *salt*-ом (*salt – со*). ***Salt*** је насумично генерисана 512-битна вредност која се чува **заједно** са хеш-ом у бази података. Ово **спречава *rainbow table* нападе** (*rainbow table* – табеле честих лозинки у хешованом формату) и обезбеђује да иста лозинка има **различите хеш** вредности за **различите кориснике**.

### JWT Token аутентикација

**JSON Web Token** (JWT) технологија је коришћена за stateless аутентикацију [15]. После успешне пријаве, сервис за управљање корисника генерише JWT токен који садржи *claims* о кориснику (*UserId, Username, Role*). **Токен је** **потписан тајним кључем који је познат само *backend* сервисима.**

JWT токен има дефинисан период важења (60 минута) после чега корисник мора поново да се пријави. Ово ограничава временски прозор за потенцијалну злоупотребу украденог токена. Потенцијално унапређење овог приступа је коришћење токена за освеживање (*Refresh Token*) који би се уместо поновног пријављивања корисника заменио за други валидни токен.

### Role-Based Access Control - Приступ на основу улога

Систем имплементира *RBAC* са две улоге:

* **User** - Стандардни корисник који може да креира квизове, решава квизове и прегледа своје резултате
* **Admin** - Администратор са свим *User* привилегијама плус могућност управљања корисницима и промовисања корисника у администраторе

Улога је укључена у JWT токен као *claim* и проверава се на сваком захтеву који захтева специфичне привилегије.

### Валидација и чишћење уноса и параметара

Сви кориснички уноси се валидирају на *backend* страни пре обраде. *UserValidationService* проверава:

* Дужину и формат корисничког имена (3-50 карактера)
* Валидност имејл адресе
* Јачину лозинке (минимум 6 карактера)
* Јединственост корисничког имена и имејла

### Заштита од честих претњи

* **SQL Injection** - *Entity Framework* користи параметризоване упите што спречава *SQL injection* нападе
* **XSS (Cross-Site Scripting)** - *React* аутоматски очисти све вредности приликом render-а
* **CSRF (Cross-Site Request Forgery)** - JWT токени у *Authorization* header-у не подлежу CSRF нападима

## Deployment и скалабилност

**Docker Compose** оркестрација омогућава једноставан *deployment* целокупног система. Једном командом *docker-compose up --build* покреће се:

* Три *SQL Server 2022* базе података (*user-db, quiz-db, result-db*)
* Три .NET 8.0 микросервиса (*User, Quiz, Result*)
* Један *Ocelot API Gateway*
* Један *React frontend* контејнер са *Nginx*

### Умрежавање контејнера

Сви контејнери су повезани на *quizhub-network* *bridge* мрежу. Ово омогућава:

* Изолацију од спољне мреже - контејнери комуницирају међусобно али нису директно доступни споља
* DNS-базирано откривање имена - контејнери се могу пронаћи по имену
* Мапирање портова - само *Gateway* (5004) и *Frontend* (3000) су доступни за комуникацију према *host* машини

### Перзистенција података кроз Volume-e

*Docker volumes* обезбеђују перзистенцију базе података. Чак и када се контејнери заустављају или рестартују, подаци остају сачувани на *host* машини.

### Мониторинг здравља сервиса

Сви сервиси имплементирају *health check* (здравствено стање) приступне тачке који враћају статус сервиса. Gateway има централизовану */health* тачку приступа која проверава статус свих *downstream* сервиса.

### Хоризонтално скалирање

Архитектура омогућава хоризонтално скалирање:

* Додавањем нових инстанци микросервиса према оптерећењу
* *Load balancing* кроз *Ocelot Gateway*
* Независно скалирање база података

За продукционо окружење, препоручује се коришћење ***Kubernetes*** оркестрације која обезбеђује аутоматско скалирање, self-healing и advanced networking могућности.

# Закључак

**QuizHub** пројекат успешно демонстрира примену микросервисне архитектуре у развоју образовних платформи. Кроз имплементацију четири специјализована сервиса (*User, Quiz, Result, Gateway*) остварена је модуларна, скалабилна и одржива архитектура која задовољава све постављене функционалне и нефункционалне захтеве.

**Кључни достигнућа пројекта:**

1. **Архитектурна елегантност** - *Database-per-Service* образац обезбеђује потпуну изолацију података између сервиса и омогућава независно скалирање база података
2. **Технолошка модерност** - Коришћење најновијих верзија *React* *19.1.1* и .*NET 8.0* гарантује дугорочну подршку и одличне перформансе
3. **Безбедносна робусност** - Вишеслојна заштита кроз JWT токене, *HMACSHA512* хеширање и *RBAC* систем обезбеђује висок ниво сигурности
4. **DevOps зрелост** - Потпуна контејнеризација омогућава лако евентуално мигрирање на комплекснија решења попут *Kubernetes-*a.

**Перформансне карактеристике:**

* Паралелно извршавање независних сервиса без блокирања
* Могућност хоризонталног скалирања према оптерећењу
* Ефикасно управљање ресурсима кроз *Docker* контејнере
* *Resilience patterns* за робусну међусервисну комуникацију

**Предлози за даља унапређења:**

1. **Event-Driven Architecture** - Имплементација асинхроне комуникације између сервиса кроз message queue системе (RabbitMQ, Kafka) за боље перформансе и *decoupling* (разлагање)
2. **Distributed Caching** - Интеграција *Redis*-а за кеширање често коришћених података као што су информације о квизовима и корисницима
3. **Kubernetes Orchestration** - Миграција на *Kubernetes* за продукционо окружење са аутоматским скалирањем и *self-healing* могућностима
4. **CI/CD Pipeline** - Аутоматизација *deployment* процеса кроз *GitHub/GitLab pipeline*-ове или *Azure DevOps* са аутоматским тестирањем

Пројекат представља солидну основу за даљи развој и може служити као референтна имплементација за сличне образовне платформе које захтевају високу скалабилност и модуларност. Микросервисна архитектура је показала своје предности у виду независног развоја, *deployment*-а и скалирања појединачних компоненти система.

# Literatura

[1] Meta Open Source. "React – A JavaScript library for building user interfaces." *Facebook*, 2023. [Online]. Доступно: https://reactjs.org/ [Приступљено: 20. септембар 2025].

[2] Microsoft Corporation. "TypeScript - JavaScript With Syntax For Types." *Microsoft*, 2023. [Online]. Доступно: https://www.typescriptlang.org/ [Приступљено: 20. септембар 2025].

[3] Hampton Catlin, Natalie Weizenbaum, Chris Eppstein. "Sass: Syntactically Awesome Style Sheets." *Sass Team*, 2023. [Online]. Доступно: https://sass-lang.com/ [Приступљено: 20. септембар 2025].

[4] Remix Software Inc. "React Router: Declarative routing for React." *Remix*, 2023. [Online]. Доступно: https://reactrouter.com/ [Приступљено: 20. септембар 2025].

[5] Recharts Team. "Recharts - A composable charting library built on React components." *Recharts*, 2023. [Online]. Доступно: https://recharts.org/ [Приступљено: 20. септембар 2025].

[6] Microsoft Corporation. ".NET 8.0 Documentation." *Microsoft Docs*, 2023. [Online]. Доступно: https://docs.microsoft.com/en-us/dotnet/ [Приступљено: 20. септембар 2025].

[7] S. Smith, "Building Microservices with .NET Core 2.0 - Second Edition," *Packt Publishing*, 2018.

[8] Microsoft Corporation. ".NET Core Guide." *Microsoft Docs*, 2023. [Online]. Доступно: https://docs.microsoft.com/en-us/dotnet/core/ [Приступљено: 20. септембар 2025].

[9] Microsoft Corporation. "SQL Server 2022 Documentation." *Microsoft Docs*, 2023. [Online]. Доступно: https://docs.microsoft.com/en-us/sql/ [Приступљено: 20. септембар 2025].

[10] Microsoft Corporation. "Entity Framework Core Documentation." *Microsoft Docs*, 2023. [Online]. Доступно: https://docs.microsoft.com/en-us/ef/core/ [Приступљено: 20. септембар 2025].

[11] Docker Inc. "Docker Documentation." *Docker*, 2023. [Online]. Доступно: https://docs.docker.com/ [Приступљено: 20. септембар 2025].

[12] T. Watson, "Ocelot Documentation." *Ocelot*, 2023. [Online]. Доступно: https://ocelot.readthedocs.io/ [Приступљено: 20. септембар 2025].

[13] Serilog Community. "Serilog - Simple .NET logging with fully-structured events." *Serilog*, 2023. [Online]. Доступно: https://serilog.net/ [Приступљено: 20. септембар 2025].

[14] Microsoft Corporation. "Dependency injection in .NET." *Microsoft Docs*, 2023. [Online]. Доступно: https://docs.microsoft.com/en-us/dotnet/core/extensions/dependency-injection [Приступљено: 20. септембар 2025].

[15] M. Jones, J. Bradley, N. Sakimura, "JSON Web Token (JWT)," *RFC 7519*, мај 2015. [Online]. Доступно: https://tools.ietf.org/html/rfc7519 [Приступљено: 20. септембар 2025].

**Пројектни фајлови (доступни на GitHub репозиторијуму):**

[16] В. Петковић, "QuizHub README.md," *QuizHub Project*, 2025. [Локални фајл]. Доступно: ./README.md

[17] В. Петковић, "Docker Compose конфигурација," *QuizHub Project*, 2025. [Локални фајл]. Доступно: ./docker-compose.yml

[18] В. Петковић, "Gateway Program.cs," *QuizHub Project*, 2025. [Локални фајл]. Доступно: ./Services/Gateway/Gateway.Api/Program.cs

[19] В. Петковић, "User Service имплементација," *QuizHub Project*, 2025. [Локални фајл]. Доступно: ./Services/UserService/UserService.Api/Services/UserService/UserService.cs

[20] В. Петковић, "Quiz Domain модел," *QuizHub Project*, 2025. [Локални фајл]. Доступно: ./Services/QuizService/QuizService.Api/Domain/Entities/Quiz.cs

Додатни линкови:

GitHub репозиторијум: <https://github.com/SaladVlad/QuizHub>