9. Nunes W., Vellasco M., Tanscheit R. Quantum-inspired evolutionary multi-objective fuzzy classifier with real and categorical representation. Journal of Intelligent & Fuzzy Systems. 2019. Vol. 36. P. 5875-5887. DOI: 10.3233/JIFS-181710.
10. Subramanian K., Savitha R., Suresh S. A complex-valued neuro-fuzzy inference system and its learning mechanism. Neurocomputing. 2014. Vol. 123. P. 110-120. DOI: 10.1016/j.neucom.2013.06.009.
11. Zhang Y., Liu Y., Li Q., Tiwari P., Wang B., Li Y., Pandey H. M., Zhang P., Song D. CFN: A Complex-valued Fuzzy Network for Sarcasm Detection in Conversations. IEEE Transactions on Fuzzy Systems. 2021. DOI: 10.1109/TFUZZ.2021.3072492.
12. Wu S.-Y., Li R.-Z., Song Y.-Q., Qin S.-J., Wen Q.-Y., Gao F. A Hierarchical Fused Quantum Fuzzy Neural Network for Image Classification. arXiv. 2024. DOI: 10.48550/arXiv.2403.09318.
13. Yao J., Guo Y. HQFNN: A Compact Quantum-Fuzzy Neural Network for Accurate Image Classification. arXiv. 2025. DOI: 10.48550/arXiv.2506.11146.
14. Satır E., Baser E. Optimization of Interval Type-2 Fuzzy Logic Controller Using Real-Coded Quantum Clonal Selection Algorithm. Elektronika ir Elektrotechnika. 2022. Vol. 28, No. 3. DOI: 10.5755/j02.eie.31148.

MOBILE PLATFORM FOR VIRTUAL CONSULTATIONS WITH DOCTORS

Bredikhin Volodymyr
phd, associate professor
Department Information system
Simon Kuznets Kharkiv National University of Economics

Abstract: The paper considers the development of a mobile platform for virtual consultations with doctors. The system architecture based on the microservice approach and H5 web technologies is proposed. The modules for searching for doctors, electronic consultations, generating electronic prescriptions, and managing medical information are implemented. The analysis of architectural solutions is carried out and the advantages of using the microservice approach for telemedical services are assessed. The results of the study confirm the possibility of creating a scalable and secure system of remote medical interaction.

Key words: telemedicine, mobile platform, microservice architecture, web application, electronic prescription, remote consultation, healthcare information system.

Introduction: With the rapid development of mobile Internet and cloud computing technology, the traditional medical service model is facing profound changes. The problems of patients with medical difficulties, uneven distribution of medical resources, low efficiency of offline clinics, etc. are becoming more and more

prominent, and there is an urgent need to build an efficient, convenient, and scalable online medical service platform through informatization.

Online consultation systems represent one of the most important components of modern telemedicine, enabling healthcare providers to deliver medical services remotely while improving accessibility and reducing operational costs.

With the rapid advancement of mobile internet and cloud computing technologies, traditional healthcare delivery models are undergoing profound transformation. Challenges such as limited patient access to care, uneven distribution of medical resources, and low efficiency in offline consultations have become increasingly prominent — urgently calling for the development of intelligent, scalable, and user-friendly online healthcare platforms through digital innovation. As a core component of the "Internet + Healthcare" ecosystem, online doctor consultation systems not only alleviate pressure on physical medical institutions but also enable patients to access high-quality medical services across geographic boundaries and around the clock. These systems demonstrate significant social value and possess broad application potential in modern digital healthcare.

In recent years, telemedicine platforms have evolved from experimental solutions into large-scale healthcare ecosystems that support consultations, electronic prescriptions, health monitoring, and remote patient management.

Despite their widespread adoption, many existing platforms focus on specific business domains such as insurance services or pharmaceutical sales, which may limit the flexibility of consultation-centered healthcare workflows. JD Health, built upon JD's e-commerce ecosystem, focuses on a "pharmaceutical retail + light consultation" model, using consultations primarily as traffic funnels for medicine sales, with most doctors being part-timers lacking systematic diagnostic support. Although both lead in user scale and capital operations, neither treats "online consultation" as an independent, core, closed-loop service module for in-depth optimization.

In contrast, this system is explicitly designed around the vertical scenario of "online consultation as the core, with medicine purchasing guided by consultation outcomes." It does not rely on insurance reimbursement logic nor prioritize pharmaceutical sales as its business focus. Instead, it generates personalized e-prescriptions through standardized consultation workflows, structured symptom collection, and professional physician judgment — guiding users to complete compliant medicine purchases based on prescriptions. This model more closely mirrors real-world medical practice, strengthens the professional linkage between "diagnosis" and "prescription," and avoids the structural imbalance of "commerce over care" prevalent in current mainstream platforms.

From the perspective of technical architecture, they rely heavily on the development of native applications, that is, they require users to download and install separate applications, resulting in poor cross platform compatibility, high maintenance costs, and poor support for low memory or elderly user devices, which creates a fixed accessibility barrier.

The purpose and objectives of the study is to develop a mobile platform for remote consultations of doctors using microservice architecture and modern web technologies.

To achieve this goal, the proposed system was designed according to several key architectural and functional requirements:

- to design the system architecture;
- to implement the main functional modules;
- to ensure the protection of medical data;
- to evaluate the advantages of the proposed approach.

The system targets the following technical objectives:

a) Build an Elastic and Scalable Microservices Architecture

Built upon Spring Boot [4], this lightweight and highly cohesive technical architecture enhances system elasticity and horizontal scalability through modular design, asynchronous processing, and caching mechanisms. It supports high-concurrency workloads and flexible deployment, meeting the performance and maintainability requirements of modern web applications.

b) Deliver a Consistent and Efficient Cross-Platform User Experience

Research results and their discussion. Developed using Vue 3 and Vant 4 [14], the responsive H5 front-end fully adapts to iOS, Android, and desktop browsers. This eliminates the cost and complexity of native app development and maintenance, lowers user adoption barriers, and enhances accessibility and convenience.

c) Support End-to-End Closed-Loop Business Processes

The system covers key business functions including online appointment scheduling, doctor consultation management, electronic medical record entry, e-prescription generation, order payment, and status synchronization — forming a complete end-to-end online medical service workflow.

d) Guarantee System Security and Data Compliance

Implement security mechanisms such as JWT-based authentication to safeguard patient privacy and ensure the security of medical data, in compliance with fundamental regulatory requirements of the healthcare industry.

Based on the identified needs of telemedicine services, the platform was designed to support online consultations, electronic prescriptions, healthcare information management, and secure communication between patients and physicians. This chapter focuses on the core objective of the “Online Doctor Consultation System Based on Microservices Architecture and H5 Technology” — to build a comprehensive, intelligent healthcare service platform that supports:

1) Online patient consultations
2) Real-time physician response
3) Electronic prescription issuance
4) Medicine delivery logistics
5) Disease knowledge dissemination

To achieve this, a systematic analysis is conducted across three dimensions: functional requirements, software layered architecture, and performance requirements, providing a clear and quantifiable foundation for subsequent system design

Administrative functions include user verification, content moderation, system monitoring, and management of healthcare information resources. Administrators review and verify physician credential submissions (e.g., medical licenses, professional certifications) prior to granting consultation privileges. They possess the authority to freeze or reactivate user accounts in response to policy violations, fraudulent behavior, or user complaints. Additionally, administrators monitor and analyze user engagement dashboards — including metrics such as Daily Active Users (DAU), consultation volume, and retention rates — to inform operational decisions and resource allocation.

All health education articles authored by physicians undergo mandatory editorial review before publication to ensure medical accuracy, appropriateness, and alignment with platform guidelines. Administrators curate and maintain the disease knowledge base (Disease Wiki), updating terminology, correcting misinformation, and organizing content by clinical relevance. Furthermore, they investigate and resolve user-reported content violations — including spam, misinformation, or inappropriate language — through content removal, user warnings, or account penalties as warranted.

To meet the system's requirements for high cohesion, low coupling, scalability, and maintainability, this system adopts a classic four-tier architecture model: Presentation Layer, Business Logic Layer, Data Access Layer, and Data Storage Layer. Each layer has clearly defined responsibilities and communicates through well-defined interfaces, facilitating independent development, testing, and deployment.

The presentation layer is responsible for user interaction and interface rendering. It is implemented using the H5 + Vue.js + Vant UI technology stack, ensuring cross-platform compatibility across mobile browsers, WeChat Mini Programs, and embedded WebViews within native apps.

Technology Stack has Framework: Vue 3.x, leveraging the Composition API [17] and <script setup> syntax for enhanced logic organization and type safety.

UI Component Library: Vant UI [14] — a lightweight, mobile-optimized component library that ensures consistent design and high performance on small screens.

State Management: Pinia [18] — a modern, type-safe state management solution for Vue, replacing Vuex with simpler APIs and better TypeScript support.

Routing: Vue Router 4.x [11] — enables dynamic, nested, and lazy-loaded routing for complex SPA navigation.

HTTP Client: Axios [12] — supports interceptors for request/response handling, cancellation tokens for aborting pending requests, and automatic JSON parsing.

Build Tool: Vite [20] — delivers near-instantaneous cold server startup and hot module replacement (HMR), significantly accelerating the development cycle.

The data access layer is responsible for interacting with the database, shielding the underlying data source differences, and providing a unified data operation interface. Mybatis plus is used as ORM framework to simplify crud operation.

Technology selection has ORM framework: mybatis plus 3.5.x [3] (enhanced version of mybatis with built-in crud, paging, and conditional constructor)

Database connection pool: hikaricp (high performance, springboot default integration) [20]

Cache integration: Spring cache [8]

Core competencies has automatically generate mapper interface and XML (through code generator) lambda expressions build query conditions (avoid hard coding field names)

Paging plug-in build on paginationinterceptor, which supports physical paging

Optimistic lock: @version annotation to prevent concurrent update conflicts

Logical deletion: @tablelogic, avoid physical deletion of data

Technical feasibility assesses whether existing technologies can support the system in achieving its intended objectives. The technology stack adopted in this system consists entirely of mature, industry-proven solutions, with no significant technical barriers identified.

The system is architecturally divided into two primary components: Client-Side (Frontend) and Server-Side (Backend), as illustrated in Figure 1.

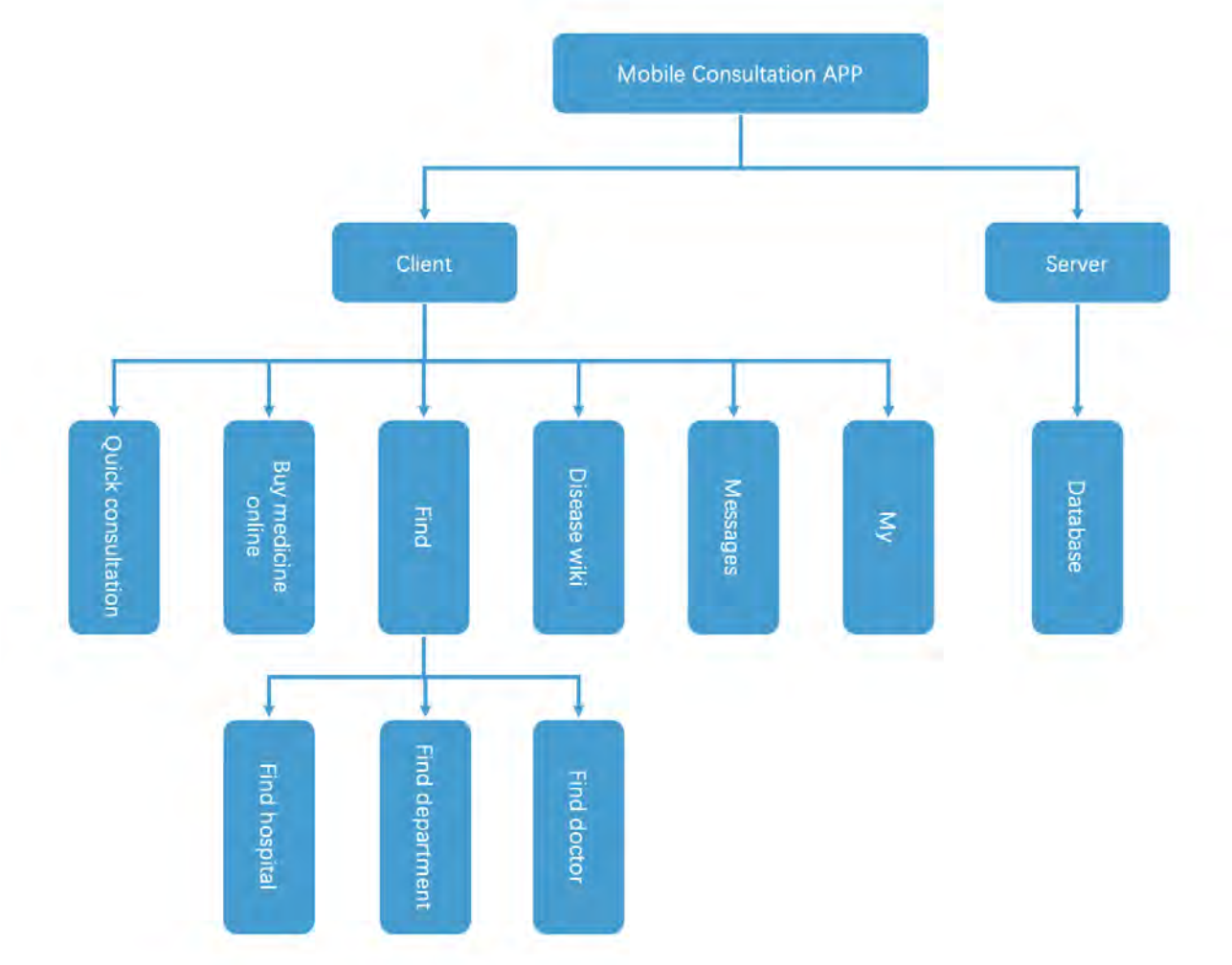

Figure 1. System Architecture Diagram

The client-side is designed around user-centric modules, enabling users to access specific functionalities based on their needs — such as searching for doctors, initiating consultations, or browsing medical knowledge. Each module provides an intuitive, responsive interface optimized for cross-platform access (mobile browsers, desktop, and embedded WebViews).

Figure 2 illustrates the proposed container-level architecture of the telemedicine platform and the interaction between its main software components

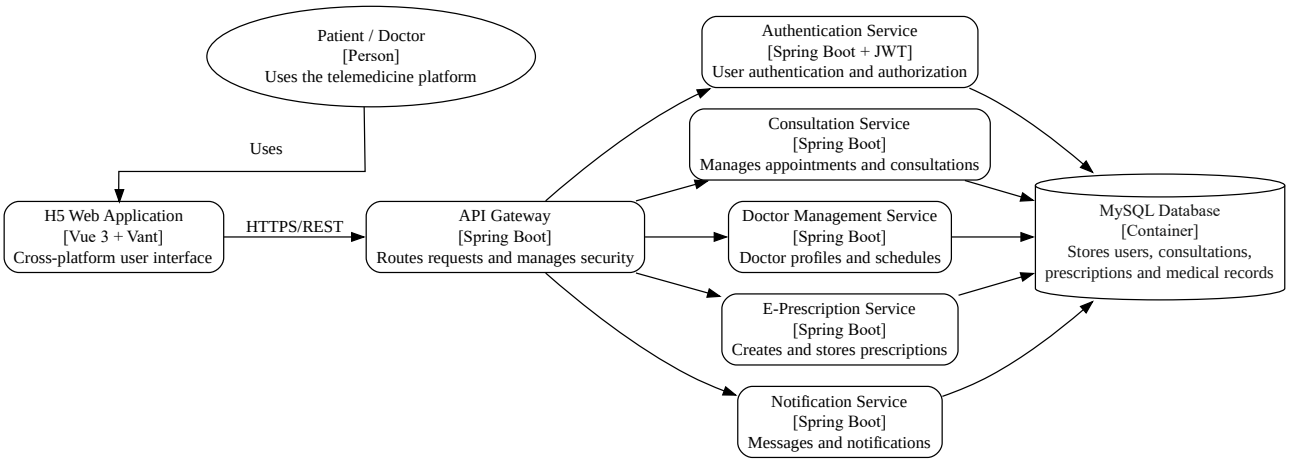

Figure 2. Container Diagram of the Telemedicine Platform Architecture

The presented architecture separates business functionality into independent services responsible for authentication, consultation management, prescription processing, and user notifications. Such separation improves scalability and simplifies system maintenance.

The proposed telemedicine platform is implemented using a microservice-based architecture that separates user interaction, authentication, consultation management, and prescription processing into independent services. Such decomposition improves scalability, maintainability, and fault isolation

The server-side exposes a suite of RESTful APIs to support client functionalities. These APIs handle core business logic and data operations, including but not limited to:

1. Hospital, department, and physician information retrieval
2. Consultation scheduling and management
3. E-prescription processing and medication ordering
4. Real-time messaging and notification delivery
5. User profile and order management

Below is a detailed description of each core functional module:

Technical Architecture Overview

This system adopts a front-end and back-end decoupled architecture. The front-end is built with Vue 3 [1] and Vant UI [2] to deliver a responsive H5 interface compatible across multiple browsers and devices, lowering user adoption barriers and maintenance costs. The back-end centers on Spring Boot with embedded Tomcat and

integrates MyBatis-Plus for efficient MySQL database [21] operations. It implements modular lightweight microservice decomposition such as User Service Appointment Service and Consultation Service to enhance system cohesion and horizontal scalability.

The architecture features clearly defined layers and responsibilities where the front-end focuses on user interaction and the back-end handles business logic and data persistence with components communicating efficiently via RESTful APIs. This design accelerates development iteration simplifies deployment and operations ensures stability under high concurrency and enables flexible future expansion. It provides a high-performance highly available and easily maintainable technical foundation tailored for online healthcare scenarios fully addressing modern web applications core demands for elasticity security and cross-platform compatibility. Refer to Architecture Figure 3.

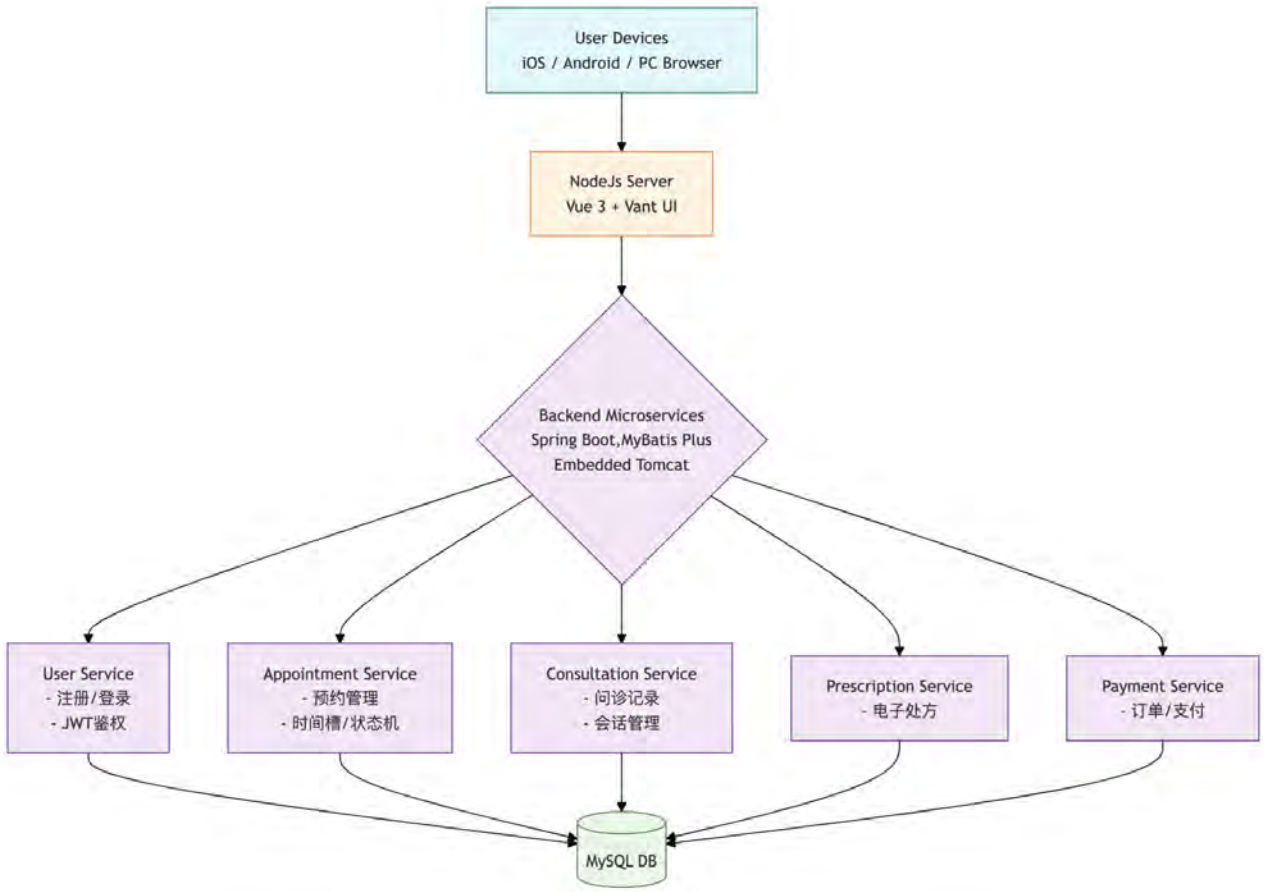

Figure 3. Technical Architecture Diagram

Users navigate to the hospital search page by clicking the “Find Hospital” button on the homepage. The page displays a search bar (Vant Search component) at the top, supporting input clearing and auto-focus.

Users enter keywords (e.g., “Beijing United Hospital”). The frontend applies a debounce mechanism (300ms) before invoking the backend search API. The backend

performs fuzzy matching using MySQL LIKE '%keyword%' or full-text indexing on the hospital name field, returning paginated results (default: 10 items per page).

Matching hospitals are displayed as a list, with each entry showing:

1. Hospital name
2. Grade (e.g., “Tertiary A”)
3. City location

If no matches are found, a message “No related hospitals found” is displayed.

Users navigate to the department search page by clicking the “Find Department” button on the homepage. The page displays a search bar (implemented using the Vant Search component) at the top, supporting real-time input, clearing, and auto-focus functionality.

Users enter keywords (e.g., “Cardiology”). The frontend applies a debounce mechanism (300ms delay) before invoking the backend search API. The backend performs fuzzy matching using MySQL LIKE '%keyword%' on the department name field and returns paginated results (default: 10 items per page).

Matching departments are displayed as a vertical list. Each entry shows the department name only (e.g., “Cardiology”, “Pediatrics”, “Dermatology”). If no matches are found, the system displays the message: “No related departments found.”

After successful login, users can initiate a physician search via the “Find Doctor” module. The system supports fuzzy search by physician name. Upon retrieving relevant results, users may click on a specific physician to view their detailed profile.

The complete workflow is illustrated in Figure 4.

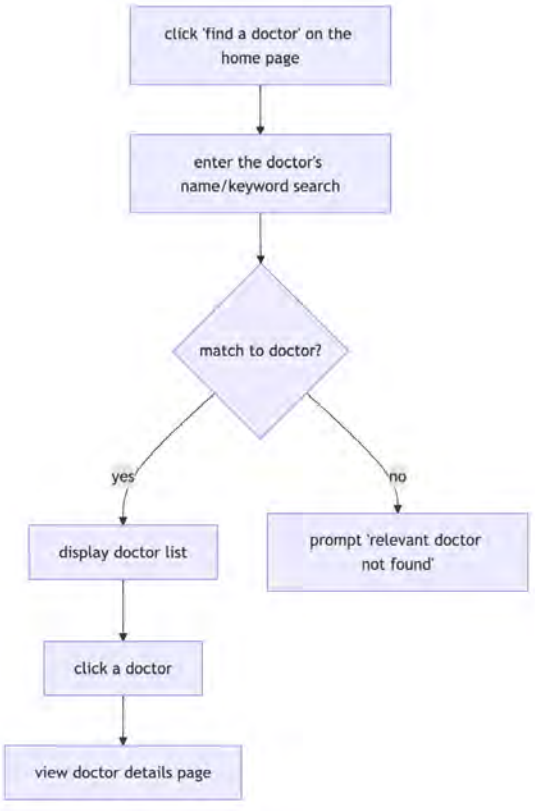

Figure 4. Find Doctor Workflow Diagram

Workflow Description

The find doctor process consists of the following steps:

Users navigate to the physician search page by clicking the “Find Doctor” button on the homepage. The page displays a search bar (implemented using the Vant Search component) at the top, supporting real-time input, clearing, and cancellation.

Users enter keywords (e.g., “Li Na” or “Pediatrics Dr. Wang”). The frontend applies a debounce mechanism (300ms delay) before invoking the backend search API. The backend performs a joint fuzzy match using MySQL LIKE '%keyword%' across multiple fields — including physician name, affiliated hospital, department, and areas of expertise — and returns paginated results (default: 10 items per page).

Matching physicians are displayed as a card-based list. Each card includes:

1. Physician avatar
2. Full name
3. Professional title (e.g., “Chief Physician”)
4. Affiliated hospital and department
5. Patient rating (e.g., 4.9/5.0)
6. Total consultation volume
7. Descriptive tags (e.g., “Fast Response”, “Expert in Pediatric Fever”)

If no matches are found, the system displays: “No related doctors found.”

Upon clicking a physician card, the user is redirected to the physician’s detailed profile page, which includes:

Basic Information – Name, avatar, title, educational background, affiliated institution, department, clinic address, and working hours.

Professional Tags – Specialized conditions treated and therapeutic approaches.

Action Buttons:

“Consult Now” – redirects to the quick consultation booking flow.

“Bookmark Doctor” – adds the physician to the user’s favorites list for future reference.

The system centers around the Quick Consultation module and integrates six core functional areas: Online Medicine Purchase, Find Hospital/Department/Doctor, Disease Wiki, Message Center, and Personal Center. Together, they form a closed-loop service covering consultation initiation, physician response, e-prescription issuance, medication purchase, health knowledge access, and user profile management.

After implementing the core functional modules and validating the system architecture, a series of experimental tests was conducted to evaluate the platform's performance, scalability, and reliability under different workloads.

Performance Evaluation and Testing Results

To evaluate the effectiveness of the proposed telemedicine platform, a series of functional and performance tests were conducted. The testing environment included a Spring Boot-based backend, MySQL database, and a Vue 3 web client deployed on a cloud server with 4 CPU cores and 8 GB RAM.

The obtained results demonstrate that the developed system provides stable operation under moderate and high workloads. During load testing, the platform

successfully handled up to 500 concurrent users while maintaining an average response time below 500 ms. The implemented microservice architecture and caching mechanisms contributed to efficient resource utilization and scalability. Security testing confirmed correct operation of JWT-based authentication and role-based access control mechanisms. Functional testing verified the correctness of all major business processes, including doctor search, appointment scheduling, online consultations, and e-prescription generation.

Table 1 – System Performance Metrics

Metric	Result
Average API response time	320 ms
Maximum API response time	780 ms
Simultaneous active users supported	500 users
Successful request rate	99.4%
System availability during testing	99.8%
Average CPU utilization	62%
Average memory utilization	68%
Database query response time	95 ms
Authentication processing time	45 ms
Functional test success rate	100%

*data generated by the author

To further illustrate the scalability characteristics of the proposed solution, Figure 5 presents the relationship between the number of concurrent users and the average API response time, fig.5..

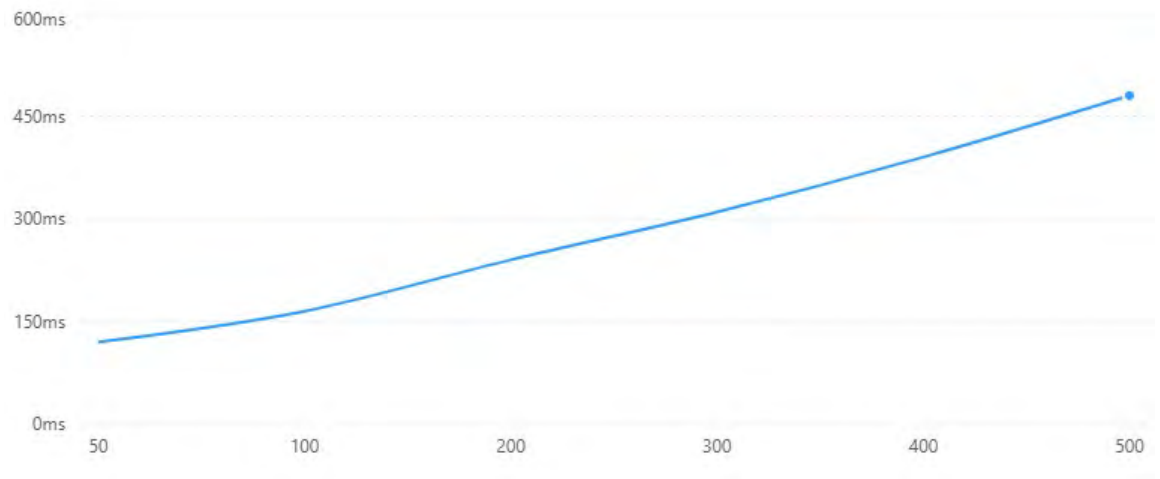

Figure 5. Response Time under Different Workloads

The results indicate that the proposed architecture is capable of supporting real-time medical consultations while maintaining acceptable performance and reliability levels. The average response time remained below 400 ms even when the number of concurrent users exceeded 400, demonstrating the scalability advantages of the microservice-based design.

Conclusions The study presented the development of a telemedicine platform based on a microservice architecture and modern web technologies. The proposed solution provides secure user authentication, online consultation management, electronic prescription generation, and cross-platform accessibility. Experimental evaluation demonstrated stable operation under workloads of up to 500 concurrent users while maintaining acceptable response times. The obtained results confirm the feasibility of applying microservice-based architectures for scalable telemedicine systems and provide a foundation for future integration of intelligent decision-support and healthcare analytics services.

List of sources used

1. Richardson, C. (2018). Microservices patterns: With examples in Java. Manning Publications.
2. Fowler, M., & Lewis, J. (2014). Microservices: A definition of this new architectural term. Martin Fowler. https://martinfowler.com/articles/microservices.html
3. VMware. (2026). Spring Boot reference documentation. Spring. https://docs.spring.io/spring-boot/docs/current/reference/html/
4. Walls, C. (2016). Spring Boot in action*. Manning Publications.
5. Newman, S. (2021). Building microservices: Designing fine-grained systems (2nd ed.). O'Reilly Media.
6. VMware. (2026). Embedded web servers. Spring Documentation. https://docs.spring.io/spring-boot/how-to/webserver.html
7. VMware. (2026). Spring beans introduction. Spring Framework Documentation. https://docs.spring.io/spring-framework/reference/core/beans/introduction.html
8. Baeldung. (2026). Spring Boot tutorials. https://www.baeldung.com/spring-boot
9. Johnson, R. (2002). Expert one-on-one J2EE design and development. Wiley.
10. Spring Project Contributors. (2026). Spring Boot. GitHub. https://github.com/spring-projects/spring-boot
11. Vue Mastery. (2026). Vue 3 Composition API guide. https://www.vuemastery.com/courses/vue-3-essentials/
12. Posva, E. (2026). Pinia: The Vue store that you will enjoy using. https://pinia.vuejs.org/
13. Microsoft. (2026). Visual Studio Code documentation. https://code.visualstudio.com/
14. Vite Team. (2026). Vite: Next generation frontend tooling. https://vitejs.dev/
15. Baomidou Contributors. (2026). MyBatis-Plus. GitHub. https://github.com/baomidou/mybatis-plus
16. Baomidou. (2026). MyBatis-Plus documentation. https://baomidou.com/en/
17. CentLinux. (2026). How to enable MySQL slow query log. https://centlinux.com/enable-mysql-slow-query-log/
18. Wooldridge, B. (2026). HikariCP: A solid, high-performance JDBC connection pool. GitHub. https://github.com/brettwooldridge/HikariCP
19. Oracle Corporation. (2026). MySQL InnoDB storage engine. MySQL Documentation. https://dev.mysql.com/doc/refman/8.0/en/innodb-storage-engine.html
20. Leith, S. (2021). High performance MySQL (4th ed.). O'Reilly Media.