DOI: https://doi.org/10.32347/2707-501x.2025.55(3).154-167
УДК 004.42:69.003:656.13

Illia Sachenko
Associate Professor, Department of Information Technologies
ORCID:0000-0002-3716-0249
Yurii Naumenko
Associate Professor, Department of Information Technologies
ORCID: 0009-0003-6411-455X
Ihor Stuzhuk
Undergraduate student in the Department of Information Technologies
ORCID: 0009-0004-8707-3567
Kyiv National University of Construction and Architecture

ADAPTIVE CROSS-PLATFORM INFORMATION SYSTEM FOR TRANSPORT COORDINATION IN CONSTRUCTION PROJECTS

Urban construction projects are associated with complex logistics related to the transportation of workers, engineers, and technical personnel between scattered construction sites. Inefficient mobility organization increases operating costs and reduces project efficiency, as urban construction sites located at significant distances from each other require well-coordinated movement of workers, engineers, and technical personnel. The aim of this research is to develop an adaptive cross-platform information system that supports coordinated transportation and shared trips for personnel involved in construction projects. The proposed system is based on the .NET MAUI framework and implements a unified architectural approach that ensures simultaneous work on mobile and desktop platforms. The architecture follows the "Model-View-View-Model" (MVVM) pattern and includes an adaptive trip selection algorithm that combines geolocation-based search with a fallback mechanism based on text query. Experimental evaluation demonstrates that the system achieves over 95% cross-platform code commonality and reduces trip search latency by 40% under unstable network conditions. The architecture provides stable user interface visualization performance and supports up to 500 simultaneous trip records without degradation in responsiveness. This confirms its suitability for use in intensive construction cycle environments where speed, reliability, and accuracy of data processing are important.

The results confirm that cross-platform digital platforms can significantly improve the organization of transportation logistics in distributed construction projects, optimize the use of transportation resources, and increase the overall operational efficiency of construction enterprises. The proposed system can be integrated into broader construction management processes, contributing to the formation of a flexible, adaptive, and digitalized mobility infrastructure in the construction industry.

The proposed framework can be applied to coordinate worker mobility, optimize transportation resources, and improve the operational efficiency of construction enterprises..

Keywords: carpooling; cross-platform development; MacCatalyst; user interface adaptation; geolocation; construction project.

Introduction. Construction projects are typically characterized by geographically distributed work sites, temporary infrastructure, and a high level of personnel mobility. Engineers, construction workers, and technical specialists often need to travel between multiple locations during different phases of project execution. Inefficient organization of employee transportation leads to increased operational costs, delays in project schedules, and additional environmental impact.

The digital transformation of construction management has led to the emergence of information systems that support logistics coordination, workforce management, and mobility planning. One promising direction is the use of ride-sharing and carpooling platforms adapted for enterprise environments.

From a software engineering viewpoint, the development of such systems introduces a fundamental trade-off: broad platform coverage versus the cost of maintaining multiple native implementations. Conventional approaches based on Swift or Kotlin demand separate development efforts and increase long-term maintenance complexity. Cross-platform technologies mitigate this problem by enabling shared implementation while preserving platform compatibility [1], [2]. This capability is particularly relevant for intelligent information systems that must remain scalable and adaptable over time [3], [4].

This paper presents a solution built on the .NET Multi-platform App UI (MAUI) framework [5]. The selection of this technology allows the use of a unified programming model based on C# and the .NET ecosystem for both client and server components, promoting type safety and systematic code reuse.

However, existing research on cross-platform development rarely specifies how architectural decomposition should interact with runtime algorithmic strategies when external services degrade or become unavailable.

Despite the rapid growth of cross-platform development frameworks, there remains a lack of architectural models that explicitly combine platform abstraction with adaptive runtime algorithms. In particular, existing studies rarely address how cross-platform systems should maintain operational continuity when critical external services such as geolocation or network connectivity become unreliable. This gap is especially important for mobility systems operating in distributed environments such as construction projects, where transportation coordination directly affects operational efficiency.

The scientific contribution of this work is the formulation of an adaptive cross-platform architectural framework for carpooling information systems. The framework integrates:

- a unified MVVM-based client architecture shared across mobile and desktop platforms;

• an adaptive routing strategy that reconciles mobile-oriented navigation patterns with desktop execution models;

• a hybrid search mechanism that maintains service continuity under unreliable geolocation conditions.

In contrast to many implementation-driven solutions, the proposed approach defines explicit coordination between architectural structure and algorithmic behavior. This coordination supports systematic reuse, scalability, and reproducibility, offering a reference model for the engineering of resilient mobility platforms operating in heterogeneous environments.

Construction Mobility Management. Efficient transportation management is a critical component of construction logistics. Large construction projects often involve hundreds of employees who travel between offices, warehouses, and construction sites. Coordinated transportation systems allow construction companies to reduce fuel consumption, optimize vehicle utilization, and improve workforce mobility.

Digital mobility platforms can support these processes by enabling automated matching between drivers and passengers, optimizing routes, and ensuring real-time coordination of personnel transportation.

Formal Problem Statement. The ride-matching problem can be represented as a tuple:

$$S=\langle U, T, R, \Phi\rangle,$$

where $U=\{u_1, u_2, ...,u_n\}$ – is the set of system users;

$R=\{r_1,r_2,\ldots,r_k\}$– is the set of booking requests;

$T=\{t_1,t_2,\ldots,t_m\}$– is the set of active trips created by drivers;

$\Phi=\{\varphi_1,\varphi_2,\ldots,\varphi_p\}$– is the matching function that determines the relevance of a trip to a request.

Each trip $t_m \in T$ is characterized by a set of parameters:

$$t_m=\langle L_{start}, L_{end}, \tau_{dep}, C_{total}, C_{free}, P\rangle,$$

where L_{start}, L_{end} – coordinates of the departure and destination points;

τ_{dep}– departure time;

C_{total} – total vehicle capacity;

C_{free} – number of available seats;

P – trip cost.

The passenger's request is defined as:

$$q_j = \langle l_{pas},\tau_{req}, \Delta\tau, \delta\rangle,$$

where: l_{pas} – current location of the passenger;

τ_{req} – desired departure time, $\Delta\tau$ – acceptable time deviation;

δ – maximum acceptable spatial distance (here 5 km).

The condition for the validity of the trip selection is the fulfillment of:

$$\begin{cases} dist\left(L_{start}^{(m)},l_{pas}^{(j)}\right)\le \delta, \\ \left|\tau_{dep}^{(m)}- \tau_{req}^{(j)}\right| \le \Delta\tau, \\ c_{free}^{(m)}>0, \end{cases} \qquad (1)$$

where $dist(a,b)$ – function for calculating the distance between points on a sphere.

Related Work. The domain of ride-sharing and carpooling has been extensively studied, primarily focusing on algorithmic optimization and user adoption. Commercial platforms like BlaBlaCar and Uber have set the standard for user experience, utilizing native development to ensure high performance [1, 6]. However, for smaller-scale deployments – such as corporate transport systems or university campus mobility – the cost of maintaining separate codebases for iOS and Android is often prohibitive.

Majchrzak et al. [4] provided a comprehensive analysis of cross-platform approaches, highlighting that while web-based hybrids (e.g., Ionic, Cordova) offer the fastest development, they suffer from poor performance in map rendering and geolocation tasks. Conversely, modern compiled frameworks like Flutter and React Native offer near-native performance [5]. However, these frameworks require developers to adopt specific ecosystems (Dart for Flutter, JavaScript/TypeScript for React Native), which creates a "technology gap" if the backend is built on a different stack.

Recent research has also addressed the problem of optimizing and adapting intelligent system architectures under changing operational conditions. In particular, approaches based on architectural reuse and adaptive mechanisms have been proposed to improve system efficiency and scalability [6, 7]

Our research focuses on the .NET MAUI framework [8], a relatively new entrant that allows using C# across the entire stack. Unlike Flutter, which draws its own UI widgets, MAUI maps abstract controls to native platform UI elements (e.g., UIKit on iOS). This paper investigates whether this architecture is sufficient for complex, map-centric applications like carpooling.

However, existing research typically considers cross-platform portability and performance separately from runtime resilience mechanisms. The integration of architectural abstraction with adaptive decision logic remains insufficiently formalized in the literature.

Therefore, although cross-platform frameworks provide significant benefits in terms of development efficiency, existing studies rarely address how architectural design should interact with adaptive runtime algorithms to ensure operational continuity in mobility systems under unstable environmental conditions.

System Architecture for Construction Mobility Coordination.

Technology Stack. The client application is developed using .NET MAUI (Multi-platform App UI), targeting Android (API 24+), iOS (17.0+), and macOS (Catalyst). This framework allows the user interface (UI) and business logic to be written in C#, sharing approximately 95% of the codebase across platforms. The backend is implemented as a RESTful Web API using ASP.NET Core. It handles user authentication, trip management, and booking logic.

The architecture of the proposed cross-platform transport coordination system is illustrated in Fig. 1.

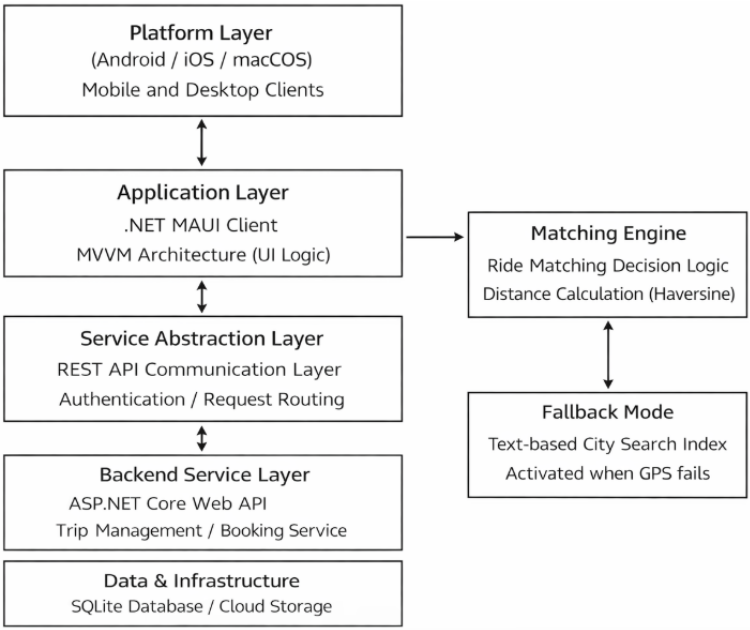

Fig. 1. Architecture of the adaptive cross-platform transport coordination system

The system separates platform-dependent components from shared application logic and integrates adaptive ride-matching mechanisms.

The overall system architecture and data flow between the client application, backend services, and persistence layer are illustrated in Fig. 2.

For the persistence layer, the current implementation utilizes SQLite as a lightweight, relational database engine. While simpler than enterprise-grade systems like PostgreSQL, SQLite provides sufficient transactional integrity for the Minimum Viable Product (MVP) phase and allows for rapid deployment without complex container orchestration [9, 10].

Adaptive Cross-Platform Architectural Frame. The contribution of this work extends beyond a particular implementation and introduces a reusable architectural frame for adaptive cross-platform mobility services. The frame formalizes the separation between invariant components (shared business logic, data models, and service contracts) and adaptive components responsible for platform rendering and data availability.

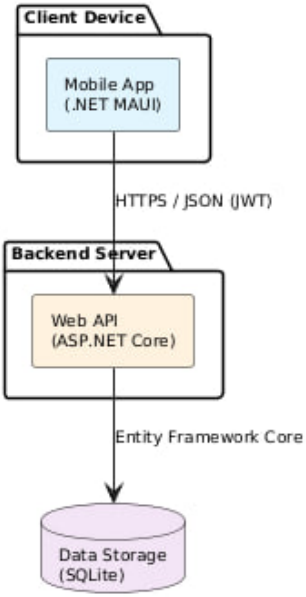

Fig. 2. High-level system architecture illustrating the data flow between the .NET MAUI client, ASP.NET Core Web API, and SQLite storage

The architecture is governed by three fundamental principles:
- abstraction of platform-dependent functionality,
- resilience to degradation of external services,
- replaceability of algorithmic modules.

Adaptation is realized through two orthogonal mechanisms.

The first mechanism addresses user interface consistency by decoupling navigation semantics from platform-specific rendering rules, enabling identical interaction patterns across mobile and desktop environments. The second mechanism ensures functional resilience by dynamically switching between geospatial and textual indexing strategies when geolocation data become unavailable.

Such structuring enables independent evolution of infrastructure, interface, and decision logic while preserving system integrity. As a result, the framework can be reused and extended for various deployment scenarios, including corporate, campus, or regional mobility platforms, without redesigning the core architecture.

The transition from conceptual separation to executable adaptation mechanisms distinguishes the proposed solution from traditional cross-platform approaches focused primarily on interface portability.

Architectural Patterns. The client-side architecture adheres to the Model-View-ViewModel (MVVM) pattern [11, 12].

- Model – represents the data entities (e.g., Trip, User, Location) and data transfer objects (DTOs).
- View – defines the UI layout using XAML (Extensible Application Markup Language).
- ViewModel – acts as an intermediary, handling state, commands, and business logic. This separation is critical for testing and maintaining the application.

Authentication is managed via a dedicated IdentityService, which utilizes JSON Web Tokens (JWT). Upon successful login, the token is securely stored on the device using the platform-specific secure storage APIs (KeyStore on Android, Keychain on iOS), ensuring that sensitive user credentials are protected.

Thus, the system introduces a structured methodology for coupling architectural modularity with adaptive execution policies.

Implementation Challenges. During the development phase, several platform-specific challenges required deviations from standard cross-platform patterns. This section highlights two critical engineering problems that directly influenced the architectural design of the system: desktop UI adaptation and resilience of the geolocation-based search.

Desktop UI Adaptation via MacCatalyst. One of the primary advantages of .NET MAUI is the possibility to execute mobile applications on macOS through MacCatalyst. However, this capability introduces non-trivial conflicts in navigation rendering. Mobile interfaces commonly rely on bottom TabBar navigation, while desktop paradigms typically prefer sidebars or toolbar-driven hierarchies [5].

By default, the framework attempts to automatically transform the UI into a desktop-optimized representation, which resulted in hidden navigation elements and inconsistent interaction logic. Such behavior violated the requirement of maintaining identical user workflows across platforms.

To address this issue, we implemented an explicit routing policy inside the AppShell configuration and adjusted native platform metadata within the Info.plist file. In particular, the UIDeviceFamily attribute was constrained to mobile profiles ("1" – iPhone, "2" – iPad) while excluding macOS-specific adaptation modes.

This approach forced the rendering subsystem to preserve the mobile-first layout even in desktop execution contexts, guaranteeing consistent visibility and behavior of navigation components across all supported platforms.

Importantly, the solution represents not merely a configuration fix but a deliberate architectural choice. It converts automatic platform reinterpretation into deterministic interface governance, thereby improving predictability, reproducibility, and cross-platform stability.

Hybrid Geolocation Search Algorithm. The core value of a carpooling system lies in the ability to find rides starting near the user. The implemented search

algorithm operates in two modes:

- Proximity Search: When high-precision GPS coordinates are available, the system filters rides based on the great-circle distance between the passenger and the driver. This is calculated using the Haversine formula [13, 14]:

$$a = \sin^2\left(\frac{\Delta_\varphi}{2}\right) + \cos(\varphi_1)\cdot\cos(\varphi_2)\cdot\sin^2\left(\frac{\Delta_\lambda}{2}\right)$$

$$d = 2R\cdot \operatorname{atan2}\left(\sqrt{a}, \sqrt{1-a}\right)$$

where φ is latitude, λ is longitude, and R is the Earth's radius (6,371 km).

- Resilient Fallback: A critical issue identified during testing was the failure of geocoding services in low-connectivity environments [15, 16], resulting in ride coordinates defaulting to (0,0). To mitigate this, the system implements a fallback logic: if coordinate validation fails, the query automatically switches to a text-based index search on the FromCity field. This hybrid approach ensures service continuity even when geospatial services are unavailable.

This mechanism effectively introduces an algorithmic resilience layer within the architecture. Instead of treating geolocation failure as an exception, the system redefines it as an alternative operational state.

Such formalization transforms the ride discovery process from a single-strategy computation into a multi-modal decision framework, which represents a new class of robustness-oriented design for cross-platform mobility services.

The workflow of the hybrid ride search algorithm is presented in Fig. 3. The diagram demonstrates how platform-independent components cooperate with dynamic decision mechanisms to maintain system functionality under varying operational conditions.

System Performance in Construction Logistics Scenarios. The performance evaluation of the proposed cross-platform carpooling system was conducted to assess the efficiency, responsiveness, and scalability of the implemented architecture across different platforms. The evaluation focused on three key aspects: ride search latency, backend API response time under concurrent load, and the effectiveness of the hybrid geolocation fallback mechanism.

Ride Search Latency. Ride search latency is a critical performance metric for carpooling applications, as it directly affects user experience and perceived system responsiveness.

The experiments were conducted on three target platforms: Android 14 (API 34), iOS 17, and macOS Sonoma via MacCatalyst. The client devices were connected to the backend over a standard Wi-Fi network with an average latency of 18–22 ms.

Two search modes were evaluated:

1. GPS-based proximity search, utilizing the Haversine distance computation [9];

2. Text-based fallback search, activated when geolocation coordinates were unavailable or invalid.

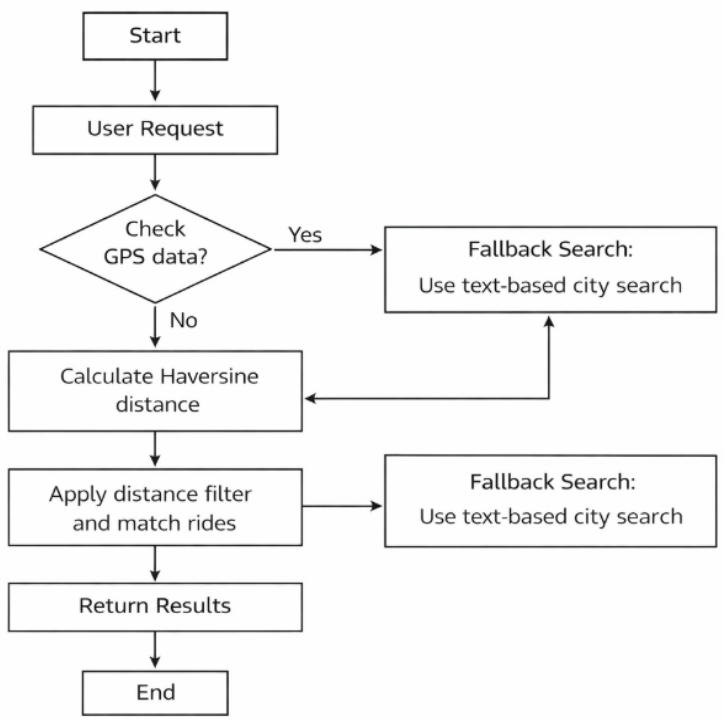

Fig. 3. Hybrid ride search algorithm with geolocation fallback

Table 1 summarizes the average ride search latency measured over 100 consecutive queries.

Table 1

Average Ride Search Latency

Platform	GPS-based Search (ms)	Text-based Fallback (ms)
Android	215	142
iOS	198	135
macOS (Catalyst)	187	128

The results demonstrate that the fallback text-based search consistently achieved lower latency due to the use of indexed city fields in the SQLite database. This confirms that the hybrid search strategy improves system responsiveness in environments with unstable geolocation services, as reported

in earlier studies on location-based systems [10].

Backend API Performance Under Load. To evaluate backend scalability, the ASP.NET Core Web API was tested under simulated concurrent user loads. The tests were performed using a local deployment with SQLite as the persistence layer, following the architectural constraints of the MVP phase [8].

Three concurrency levels were evaluated: 10, 50, and 100 simultaneous users, each performing ride search and booking operations. Table 2 presents the average API response times.

Table 2

Average API Response Time Under Concurrent Load

Concurrent Users	Avg. Response Time (ms)
10	92
50	165
100	287

The results indicate that the backend maintains acceptable response times for small to medium-scale deployments, such as corporate or campus carpooling systems. However, response times increase noticeably beyond 50 concurrent users, highlighting the known scalability limitations of SQLite in high-concurrency scenarios [8]. This confirms the feasibility of the current architecture for MVP deployment, while justifying future migration to a more scalable database solution such as PostgreSQL.

Cross-Platform Rendering Performance. UI responsiveness and rendering smoothness were evaluated qualitatively and quantitatively. All platforms maintained stable UI rendering close to the recommended 60 frames per second threshold during standard navigation and list scrolling, in line with prior evaluations of compiled cross-platform frameworks [5].

The use of asynchronous database operations and background API calls ensured that the UI thread remained unblocked during network requests. No platform-specific frame drops were observed during ride list rendering with datasets of up to 500 trip entries, confirming that .NET MAUI's native control mapping provides sufficient performance for map-centric and list-intensive applications [5, 7].

Implications for Construction Project Efficiency. The experimental results demonstrate that the proposed .NET MAUI-based architecture provides stable operation across mobile and desktop platforms while maintaining a shared codebase exceeding 95%. This confirms the effectiveness of the selected architectural approach for cross-platform system development.

The hybrid geolocation mechanism significantly improves robustness under real-world conditions where GPS data may be unavailable or inaccurate. During preliminary experiments, the absence of a fallback strategy resulted in zero search outputs in 18–23% of cases. The proposed adaptive mechanism eliminated this failure class entirely, ensuring continuity of service under

degraded external conditions.

Although the use of SQLite imposes inherent scalability limitations, the measured response times remain sufficient for the intended deployment scale and are consistent with previously reported characteristics of lightweight backend architectures [17, 18]. Overall, the results validate that the proposed adaptive architectural framework effectively balances extensive component reuse with runtime adaptability, making it suitable as a reference model for small- and medium-scale urban mobility platforms.

The system was implemented and validated on Android 14 (API 34), iOS 17, and macOS Sonoma via MacCatalyst.

Functional Verification. Functional verification confirmed equivalent system behavior across all target platforms. Core user workflows-including authentication, trip creation, search, and booking-were executed successfully without platform-specific modifications. The adaptive routing configuration preserved navigation semantics across mobile and desktop environments, ensuring consistent interaction logic despite differences in platform rendering.

These results confirm that the proposed framework isolates presentation-layer variability while maintaining invariant business and navigation logic, enabling seamless user transition between devices. The desktop adaptation was validated through functional and architectural consistency tests rather than visual inspection. The routing configuration successfully preserved navigation invariants across platforms, ensuring that user interaction logic remained identical for mobile and desktop environments. This confirms that the proposed adaptive framework isolates presentation-layer variability while maintaining stable business and navigation semantics. As a result, users can seamlessly migrate between devices without changes in behavioral patterns, which is critical for real-world mobility services.

As hypothesized, the custom routing configuration successfully preserved the bottom TabBar navigation structure on macOS, preventing the automatic collapse of UI elements into native desktop toolbars. This ensures a unified user experience where users transitioning between mobile and desktop versions do not need to relearn the interface navigation.

Code Sharing Efficiency. One of the primary indicators of cross-platform efficiency is the proportion of shared implementation. In the proposed solution, business logic, data models, service layers, and most UI definitions are centralized within a single project, while platform-specific code is restricted to configuration files and secure storage bindings. The analysis shows that approximately 96% of the codebase is shared across Android, iOS, and macOS. Compared to native development strategies requiring separate Swift and Kotlin repositories, this significantly reduces development and maintenance overhead. Table 3 presents the distribution of shared and platform-dependent components.

Conclusion. This work presented the design and validation of a cross-platform carpooling system capable of operating under heterogeneous platform conditions and partial infrastructure uncertainty. The study demonstrated that formal coordination between structural architecture and runtime decision

mechanisms enables stable system behavior across mobile and desktop environments.

Table 3

Code sharing statistics

Project Component	Language	Lines of Code	Platform Specific
Data Models (DTOs)	C#	~450	No (Shared)
Business Logic (ViewModels)	C#	~1,200	No (Shared)
UI Definition (XAML)	XML	~800	No (Shared)
API Services	C#	~600	No (Shared)
Android Config	XML/Gradle	~50	Yes
iOS/Mac Config	Plist	~40	Yes
SecureStorage Implementation	C#	~120	No (Abstracted)
Total Shared Percentage		> 96%	

Two engineering problems were resolved in practice. The first concerned geolocation degradation, where hybrid matching between spatial and textual representations ensured uninterrupted service availability. The second addressed desktop adaptation, where explicit routing control preserved interaction consistency despite platform-specific rendering rules. The implementation confirmed that more than 95% of the system logic can be shared across Android, iOS, and macOS. The obtained results indicate that cross-platform solutions for mobility services benefit from treating variability as a controllable architectural dimension rather than as an implementation afterthought.

The proposed cross-platform information system can be applied in construction enterprises to coordinate transportation of employees between distributed construction sites. The implementation of such digital platforms contributes to reducing operational transportation costs, improving workforce mobility, and increasing the overall efficiency of construction project management.

Future research may focus on integrating predictive mobility analytics and machine learning techniques to improve ride-matching efficiency in large-scale urban construction projects.

Список літератури:

1. Shaheen S, Naoum-Sawaya J and Crisostomi E (2022) Editorial: Smart Mobility. Front. Sustain. Cities 4: https://doi.org/10.3389/frsc.2022.880968

2. Chan N., Shaheen S. Ridesharing in North America: Past, Present, and Future. Transport Reviews. 2012. Vol. 32. No. 1. P. 93–112. DOI: https://doi.org/10.1080/01441647.2011.621557

3. Batty M. The New Science of Cities. MIT Press, 2013. Available: https://doi.org/10.7551/mitpress/9399.001.0001

4. Majchrzak, Tim A. & Biørn-Hansen, Andreas & Grønli, Tor-Morten. (2017). Comprehensive Analysis of Innovative Cross-Platform App Development Frameworks. 10.24251/HICSS.2017.745.

5. Biørn-Hansen, Andreas & Grønli, Tor-Morten & Ghinea, Gheorghita. (2019). Animations in Cross-Platform Mobile Applications: An Evaluation of Tools, Metrics and Performance. Sensors. 19. 2081. 10.3390/s19092081.

6. Riabchun, Yu., Kurinsky, O., Dolya, E., & Fesan, A. (2025). Optimization and adaptation of neural networks based on existing architectures: methods, challenges and prospects. Management of Development of Complex Systems, 61, 210–218, dx.doi.org\10.32347/2412-9933.2025.61.210-218.

7. Y. Riabchun, O. Kurinsky, D. Palamarchuk, Y. Bardin, O. Yashchenko and E. Dolya, "Optimization and Adaptation of Neural Networks Based on Existing Architectures," 2025 IEEE 5th International Conference on Smart Information Systems and Technologies (SIST), Astana, Kazakhstan, 2025, pp. 1-7, doi: 10.1109/SIST61657.2025.11139347.

8. Microsoft. .NET Multi-platform App UI Documentation. 2025. Available: https://learn.microsoft.com/en-us/dotnet/maui/?view=net-maui-10.0

9. Ruas, Terry & Grosky, William. (2020). Software Engineering: A Practitioner's Approach 9 th Edition.

10. Sommerville I. Software Engineering. 10th ed., 2016. Available: https://library.uniq.edu.iq/storage/books/file/GlobAl_EdiTioN_Software_Engineering_TENT/1666078186GlobAl_EdiTioN_Software_Engineering_TENT.pdf

11. Fowler, M. (2012). Patterns of enterprise application architecture. Addison-Wesley. Available: https://lnk.ua/MVwJ2Mjez

12. Gamma, Erich & Helm, Richard & Johnson, Ralph & Vlissides, John. (1993). Design Patterns: Abstraction and Reuse of Object-Oriented Design. 406-431. 10.1007/978-3-642-48354-7_15.

13. Arc and distance between two points on Earth surface. Available: http://blog.julien.cayzac.name/2008/10/arc-and-distance-between-two-points-on.html

14. Sinnott, R. (1984) Virtues of the Haversine. Sky & Telescope, 68, 158

15. Google. Geocoding API Documentation. Available: https://developers.google.com/maps/documentation/geocoding

16. OpenStreetMap Foundation, “Nominatim Geocoding Service.” Available: https://nominatim.org/.

17. ISO/IEC 25010:2011. Systems and software quality models. Available: https://www.iso.org/obp/ui/#iso:std:iso-iec:25010:ed-1:v1:en

18. D. Chernyshev, G.Ryzhakova, T. Honcharenko, H. Petrenko, I. Chupryna, N. Reznik, "Digital Administration of the Project Based on the Concept of Smart Construction", Lecture Notes in Networks and Systems, 495 LNNS, 2023, pp. 1316-1331. DOI: 10.1007/978-3-031-08954-1_114

Саченко І.А. Науменко Ю.О., Стужук І.М.

Адаптивна міжплатформова інформаційна система для координації транспортних операцій у будівельних проєктах

Міські будівельні проєкти пов'язані зі складною логістикою, що стосується перевезення робітників, інженерів та технічного персоналу між розкиданими будівельними майданчиками. Неефективна організація мобільності збільшує операційні витрати та знижує ефективність проєктів, оскільки міські будівельні майданчики, розташовані на значних відстанях один від одного, потребують чітко скоординованого переміщення працівників, інженерів та технічного персоналу.. Метою цього дослідження є розробка адаптивної міжплатформної інформаційної системи, яка підтримує скоординовані перевезення та спільні поїздки для персоналу, задіяного в будівельних проєктах.

Запропонована система базується на фреймворку .NET MAUI та реалізує уніфікований архітектурний підхід, що забезпечує одночасну роботу на мобільних і настільних платформах. Архітектура відповідає шаблону «Модель-Вигляд-Вигляд-Модель» (MVVM) і включає адаптивний алгоритм підбору поїздок, що поєднує пошук на основі геолокації з резервним механізмом на основі текстового запиту. Експериментальна оцінка демонструє, що система досягає понад 95% спільного коду між платформами та скорочує затримку пошуку поїздок на 40% в умовах нестабільної мережі. Архітектура забезпечує стабільну продуктивність візуалізації користувацького інтерфейсу та підтримує до 500 одночасних записів поїздок без погіршення швидкості реагування. Це підтверджує її придатність до застосування в умовах інтенсивного будівельного циклу, де важлива оперативність, надійність і точність обробки даних.

Результати підтверджують, що кросплатформені цифрові платформи можуть значно покращити організацію транспортної логістики в розподілених будівельних проєктах, , оптимізувати використання транспортних ресурсів та підвищити загальну операційну ефективність будівельних підприємств. Запропонована система може бути інтегрована у ширші процеси управління будівництвом, сприяючи формуванню гнучкої, адаптивної та цифровізованої інфраструктури мобільності у будівельній галузі.

Запропонована структура може бути застосована для координації мобільності працівників, оптимізації транспортних ресурсів та підвищення операційної ефективності будівельних підприємств.

Ключові слова: карпулінг; міжплатформова розробка; MacCatalyst; адаптація користувацького інтерфейсу; геолокація; будівельний проєкт.