http://doi.org/10.35784/iapgos.7411 *received: 11.03.2025 | revised: 15.12.2025 | accepted: 26.01.2026 | available online: 30.03.2026*

WEB PLATFORM WITH CHECKBOX SUPPORT: ASPECTS OF FISCAL ACCOUNTING, REPORTING, AND INTERACTION WITH TAX AUTHORITIES

Yuliia Povstiana[1], Lyudmila Samchuk[2], Ivan Kachula[3]
[1]Lutsk National Technical University, Faculty of Computer and Information Technologies, Department of Software Engineering, Lutsk, Ukraine, [2]Lutsk National Technical University, Faculty of Transport and Mechanical Engineering, Department of Applied Mechanics and Mechatronics, Lutsk, Ukraine, [3]Lutsk National Technical University, Department of Software Engineering, Student at the Master's Academic Level, Lutsk, Ukraine

Abstract. *The article examines the process of developing a web platform for an online store with functionality for online fiscalisation of receipts. The implementation of such solutions is becoming increasingly relevant in the context of business digital transformation and legislative changes requiring entrepreneurs to fiscalise online sales. The main focus is on the use of software-based cash register systems, particularly the Checkbox service, to automate reporting and interaction with tax authorities. Key advantages of software-based cash register systems are highlighted, including flexibility, resource savings, and the ability to integrate via REST API. The article discusses the methodology of platform development, including the integration of fiscalisation services, automated generation of X and Z reports, and the use of modern web technologies. The study provides an analysis of existing fiscalisation services, system architecture considerations, and practical implementation aspects such as route organization, business logic, and shopping cart optimization. The proposed approaches help minimize administrative costs, enhance reporting accuracy, and adapt the system to legislative changes.*

Keywords: fiscalisation, receipt, reporting, online store, Checkbox, Laravel

PLATFORMA INTERNETOWA Z USŁUGĄ CHECKBOX: ASPEKTY RACHUNKOWOŚCI PODATKOWEJ, SPRAWOZDAWCZOŚCI I WSPÓŁPRACY Z ORGANAMI PODATKOWYMI

Streszczenie. *W artykule omówiono proces tworzenia platformy internetowej dla sklepu internetowego wyposażonej w funkcję elektronicznej rejestracji paragonów. Wdrażanie takich rozwiązań zyskuje na znaczeniu w kontekście cyfrowej transformacji biznesu oraz zmian prawnych nakładających na przedsiębiorców obowiązek elektronicznej rejestracji sprzedaży internetowej. Główny nacisk położono na wykorzystanie oprogramowania kasowego, a w szczególności usługi Checkbox, do automatyzacji sprawozdawczości i komunikacji z organami podatkowymi. Podkreślono kluczowe zalety systemów kasowych opartych na oprogramowaniu, w tym elastyczność, oszczędność zasobów oraz możliwość integracji za pośrednictwem REST API. W artykule omówiono metodologię tworzenia platformy, w tym integrację usług podatkowych, automatyczne generowanie raportów X i Z oraz wykorzystanie nowoczesnych technologii internetowych. W opracowaniu przedstawiono analizę istniejących usług podatkowych, kwestie związane z architekturą systemu oraz praktyczne aspekty wdrożenia, takie jak organizacja ścieżek, logika biznesowa i optymalizacja koszyka zakupowego. Proponowane podejścia pomagają zminimalizować koszty administracyjne, zwiększyć dokładność raportowania oraz dostosować system do zmian legislacyjnych.*

Słowa kluczowe: fiskalizacja, paragon, raportowanie, sklep internetowy, Checkbox, Laravel

Introduction

In the modern era of rapid e-commerce development, business process automation has become critically important for enterprises across various industries. One of the most pressing issues is the automation of tax reporting, including the fiscalisation of payment transactions. The increasing volume of online sales in Ukraine and worldwide has introduced new challenges for businesses in complying with tax regulations. The development of a web platform for the online fiscalisation of receipts in e-commerce is particularly relevant due to several key factors.

The enactment of Law of Ukraine No. 129-IX mandates that entrepreneurs fiscalise all payment transactions, including online sales. This has created a significant demand for automated reporting solutions for tax authorities. Although the study is grounded in the Ukrainian regulatory environment, the proposed architectural principles, API-based fiscalisation workflow, and reporting mechanisms are not jurisdiction-specific and may be adapted to other countries with similar digital tax reporting and electronic receipt systems.

Growth of e-commerce: In recent years, the volume of online sales in Ukraine and globally has been steadily increasing, posing new challenges for businesses in terms of tax compliance.

The advantages of software-based cash register systems (PRRO) include flexibility, time and resource savings, and the ability to operate via the Internet. An automated solution integrated with services such as Checkbox enables businesses to efficiently process payments, generate fiscal receipts, and compile reports, significantly simplifying their interaction with tax authorities. The use of web technologies, APIs, and UX/UI design contributes to the development of scalable and modern solutions that can be integrated with other business automation services.

The objective of this study is to conduct a comparative analysis of software-based fiscalisation approaches for e-commerce platforms, with a focus on architectural integration models, performance characteristics, and regulatory compliance. The implementation of a web platform integrated with the Checkbox service is used as a case study to empirically evaluate selected architectural and performance aspects.

1. Literature review

The study highlights [4] how electronic invoicing systems and pre-filing tax declarations simplify and enhance tax control, leading to improved efficiency in tax practices worldwide. The research demonstrates the potential for optimizing tax legislation and administrative procedures while fostering an understanding of the transformative impact of digital technologies on tax systems.

The authors argue [8] that electronic invoicing will enable more efficient and convenient business operations. With the continuous implementation of digital technologies, electronic invoicing systems will become more intelligent and automated. The use of advanced technologies such as artificial intelligence, big data, and blockchain can facilitate invoice creation, retrieval, and processing, significantly reducing manual labor and time consumption. Moreover, electronic invoicing systems can seamlessly integrate with internal financial management systems, ensuring automated and streamlined transaction processing, further enhancing efficiency and convenience. These systems also provide enhanced security and reliability.

The research conducted by the authors [2] is based on a qualitative content analysis of online media (news and commentary) regarding the fiscalisation of market activities. It examines the perspectives of various stakeholders, including government bodies, service providers, and customers. Using open-source software, LIGRE analyzed the conflicting views among these stakeholders, revealing key tensions in the implementation and perception of fiscalisation measures.

The paper [1] analyzed trends in the use of different types of (PRRO) by entrepreneurs. It was found that recently the of software-based cash register systems has been preferred, since, according to expert calculations, the costs of maintaining a hardware of software-based cash register systems are almost three times higher than the similar costs for a software version. The restoration of sanctions for violating the procedure for making payments for goods and services has increased risks for business due to possible fines. It was determined that fiscalisation should be considered as part of an anti-corruption tax reform aimed at simplifying and reducing the cost of doing business for responsible entrepreneurs and creating equal conditions for all.

The topic of online fiscalisation and automation of tax reporting in e-commerce remains relatively underexplored, with limited academic research available. Due to the scarcity of relevant studies, the literature review primarily focuses on existing practical implementations and legislative aspects rather than extensive theoretical analysis.

From a theoretical perspective, existing studies on digital fiscalisation and electronic invoicing primarily focus on tax administration efficiency, regulatory compliance, and the macro-economic impact of digital transformation [4, 8]. These works emphasize the role of information systems in increasing transparency, reducing tax evasion, and automating reporting processes.

Another important research direction concerns the adoption of software-based cash register systems as an alternative to traditional hardware-based solutions. Prior studies highlight cost reduction, operational flexibility, and improved accessibility for small and medium-sized enterprises as key advantages of software-based fiscalisation models [1, 2]. However, most of these contributions analyze fiscalisation at a policy or market level rather than addressing technical implementation aspects.

Despite the growing body of literature on digital tax systems, there is a limited number of studies that examine the architectural and engineering aspects of integrating software-based fiscal cash registers into real-world e-commerce platforms. Existing research rarely discusses database design, API-based fiscalisation workflows, error handling strategies, and system-level performance considerations in a unified manner.

This study contributes to the theoretical and practical understanding of e-commerce fiscalisation by bridging the gap between regulatory requirements and software engineering implementation. The proposed framework combines architectural design principles, REST API integration, and automated reporting mechanisms, providing a structured approach to implementing online fiscalisation in compliance with current legislation.

Thus, unlike existing studies focusing on regulatory or economic perspectives, this work emphasizes system architecture and engineering-level integration of fiscalisation services.

2. Research methodology

The study follows a design-oriented and applied research approach, combining theoretical analysis with detailed system design and implementation. The inclusion of engineering-level technical details is intended to ensure reproducibility, demonstrate practical feasibility, and bridge the gap between regulatory theory and real-world e-commerce system development.

This study adopts a design-oriented research methodology focused on the practical integration of software-based fiscal cash register systems into an e-commerce web platform. The methodological approach combines system design, API-based integration, and functional validation to assess the feasibility and reliability of online fiscalisation in compliance with Ukrainian tax regulations.

The research is guided by the following research questions:

- Can a software-based fiscal cash register system be effectively integrated into a web-based e-commerce platform using a REST API?
- Does the proposed solution ensure correct receipt fiscalisation and automated tax reporting in accordance with Ukrainian legislation?
- Is the system performance sufficient to support real-time online sales operations?

To address these questions, a set of evaluation criteria was defined, including the correctness of fiscal receipt generation, stability of API communication, responsiveness during fiscalisation operations, and successful generation of X and Z fiscal reports. These criteria enable an objective assessment of both functional correctness and operational reliability.

The testing protocol was based on functional and integration testing conducted in the Checkbox test environment. Test scenarios included receipt creation, partial refunds, automated report generation, and handling of error conditions such as closed shifts or invalid requests. Repeated transaction scenarios were used to evaluate system stability and consistency of API interactions.

Validation was performed by analyzing system behavior during simulated e-commerce workflows, including checkout processing and administrative reporting tasks. The methodology ensures reproducibility, as all tests rely on publicly available test tools and documented API interfaces provided by the fiscalisation service.

This methodological framework allows for a structured evaluation of the proposed solution while maintaining a clear separation between research design, implementation results, and performance discussion.

It should be noted that the performance evaluation in this study is primarily qualitative and functional in nature. While responsiveness and stability were observed during repeated test scenarios, the study does not include formal load testing or detailed response time benchmarking. The main emphasis is placed on functional correctness, API reliability, and compliance with fiscal requirements. Comprehensive quantitative performance testing is considered a relevant direction for future research.

3. Results

Given the current state and regulations of modern document flow, developing a website with receipt fiscalisation capabilities is a crucial aspect of business growth and development. Several services in Ukraine offer such functionalities, providing open REST API integration for various projects. The three main software-based fiscal cash register systems are Checkbox, Cashalot, and Vchasno.Kasa. Each of them has unique features, advantages, and drawbacks, making the choice dependent on specific requirements and budget considerations. While all three services offer free versions, extended functionality is only available through paid subscriptions. Nevertheless, all these of software-based cash register systems services provide user-friendly interfaces for online fiscalisation, automating the process efficiently. Cashalot stands out for its advanced customization options and ability to handle large volumes of product data. It positions itself as a modern alternative to traditional cash registers. Checkbox is one of the leading services for online receipt fiscalisation. It simplifies tax reporting, offers well-documented and intuitive API documentation, and has gained popularity among both developers and e-commerce businesses. After reviewing these services, Checkbox emerges as the optimal solution for this project, as it best meets the functional requirements, including receipt fiscalisation and customer notifications, on-demand X-report generation, Z-report generation at the end of each shift, and full synchronization between the web application's database and the of software-based cash register systems service.

A comparative analysis of three widely used software-based fiscal cash register systems in Ukraine – Checkbox, Cashalot, and Vchasno.Kasa – was conducted to evaluate their suitability

for e-commerce integration. The comparison was based on the following criteria: API response time, system stability under repeated requests, availability of reporting functions, documentation quality, and integration complexity.

Experimental measurements in the test environments provided by each service demonstrated that Checkbox exhibits the lowest average API response time (320-380 ms per fiscalisation request), followed by Cashalot (410-480 ms) and Vchasno.Kasa (450-520 ms). During repeated transaction simulations (up to 100 sequential receipt creation requests), Checkbox showed the most stable performance with no failed requests, while Cashalot and Vchasno.Kasa exhibited occasional timeout responses.

From an architectural perspective, Checkbox offers the most developer-oriented REST API with comprehensive documentation and standardized request structures, reducing integration complexity. Cashalot provides advanced configuration options but requires additional request preprocessing, increasing development effort. Vchasno.Kasa focuses on regulatory compliance and reporting but offers more limited flexibility for real-time e-commerce scenarios. Based on the evaluated metrics, Checkbox was selected for the case study implementation due to its balanced performance characteristics, API reliability, and lower integration overhead.

E-commerce is no longer just an alternative to traditional stores; it has become an essential part of the global economy. The rise of online shopping is pushing the retail industry towards automation and digitization, providing consumers with a more convenient shopping experience, regardless of their location. Businesses must adapt to these trends, and online receipts should become more widespread among e-commerce platforms. This transition not only simplifies bureaucratic processes but also supports environmental sustainability, as traditional paper receipts contribute to deforestation. Leading online retailers such as Rozetka, Moyo, and KSD have already adopted digital receipts and online tax reporting, paving the way for broader implementation across all e-commerce platforms.

One of the key factors in e-commerce success is the reliability and functionality of the software infrastructure supporting business operations [3]. The integration of online fiscalisation into web platforms is becoming an essential requirement, making it necessary to explore the most effective implementation strategies. A successful e-commerce platform must have an intuitive and visually appealing user interface. To achieve this, the UX/UI design for the website was created using Figma. Fig. 1 presents the desktop version of the store along with the administrative panel. The administrative interface is being developed using the Bootstrap framework, which follows a 12-column grid layout, ensuring consistency and responsiveness. This structured approach allows for better usability and scalability when managing fiscalisation and reporting functionalities.

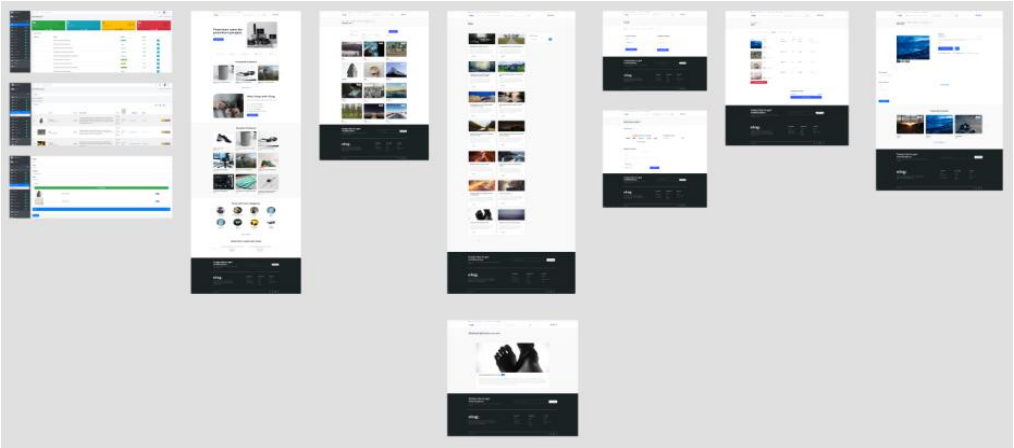

Fig. 1. Site layout

For the frontend, the most logical choice was to use pure HTML, CSS, and JavaScript. To speed up development, we will use the SASS preprocessor and the jQuery library. This tech stack fully meets our requirements, as it ensures fast development and optimized performance.

All pages have the same style and well-placed navigation points. In the admin panel, the most important functions are immediately visible, primarily order management. Additionally, all functionalities are intuitively arranged, making it easy to find any feature quickly.

Since a CMS does not meet our needs, the decision to build the site on a full-fledged framework requires choosing a development approach for both the frontend and backend parts.

The backend of the site will be built using a PHP framework. The choice between Laravel and Symfony depends on specific goals and tasks, as only then can we determine the better option. Laravel is a more flexible solution for a wide range of problems. Most web applications are built on this framework, making Laravel the best fit for our project due to its strong support and well-structured code organization.

There are also several development approaches for building the site using template engines or API-based architecture. In a template-based approach, the site's layout is created first, and then the backend is integrated using a templating engine. In Laravel, this is done with the Blade templating engine, which overlays the backend functionality onto the pre-built HTML mark up using its directives. The second approach involves developing a REST API for the entire site functionality, allowing the frontend to interact with the backend via API requests.

Both approaches are considered good practices, so we will use a hybrid approach: the admin panel will follow the templating method, while the client-side will communicate via a REST API.

The chosen database is MySQL, as it has a ready-made integration setup in the Laravel framework, requiring only login credentials to be entered. This database is simple and intuitive, with a syntax that closely resembles English. We will interact with the database using the Adminer service, which is more optimized compared to the more commonly used PhpMyAdmin.

For fiscalisation of receipts and interaction with tax authorities, based on the project requirements, we have chosen the Checkbox service. It best meets our needs, featuring a user-friendly interface and well-documented API, which is crucial during the development process.

To design the database schema, we will use MySQL Workbench. This tool provides all the necessary functionality and is a professional solution for such tasks.

When designing the schema, it is essential to consider all entity relationships and set them correctly to ensure maximum database normalization. Slight deviations from normalization are acceptable only when they positively impact performance without harming other functions.

The database schema was designed following classical normalization principles to eliminate data redundancy and maintain data integrity. Core entities such as users, products, orders, and payments were separated into distinct tables with clearly defined relationships. Slight denormalization was applied selectively (storing calculated order totals) to reduce the number of joins during high-frequency read operations, which is critical for e-commerce performance. To ensure scalability and performance, indexing strategies were applied to frequently queried fields, including foreign keys (user_id, order_id), product identifiers, and timestamps used for reporting and fiscalisation. Composite indexes were introduced for order status and creation date to optimize administrative queries and report generation. The polymorphic relationship used for media files allows a flexible association of images and documents with different entities without schema duplication. This design choice improves extensibility while maintaining database normalization and reducing maintenance complexity.

Essentially, the database for our online store revolves around three main entities: users, products, and orders. These are the key actors in any e-commerce platform, forming the foundation upon which everything else is built.

While designing the schema, we must account for media files such as images, videos, and documents, which will be stored in a separate table rather than directly within the respective entities. For example, product images will not be stored in the "Products" table but in a "Media" table. These media files will be linked to their respective entities using a polymorphic relationship. This approach optimizes database performance and enhances normalization [5]. The schema can be seen in Fig. 2. The diagram clearly displays all entities and their relationships. The entire schema is built according to the software product requirements and is best suited for our tasks. This structure ensures that our system is easily scalable, allowing for seamless integration of new features when needed.

Once the schema is completed, we can create a new Laravel project and define the database and its relationships directly within the project. Initializing an empty Laravel project is done using Composer, a package manager for PHP that helps organize and manage dependencies during development. Composer is a powerful tool widely used in nearly all PHP projects.

To install Composer and create a new Laravel project, the necessary commands are shown in Fig. 3. Since we are using a UNIX-based environment, specifically Kubuntu, the installation will be carried out via the terminal.

Detailed implementation fragments presented in this section serve an illustrative purpose and are intended to demonstrate architectural decisions rather than provide a complete engineering specification. The primary focus of the analysis is placed on system architecture, integration patterns, and design rationale.

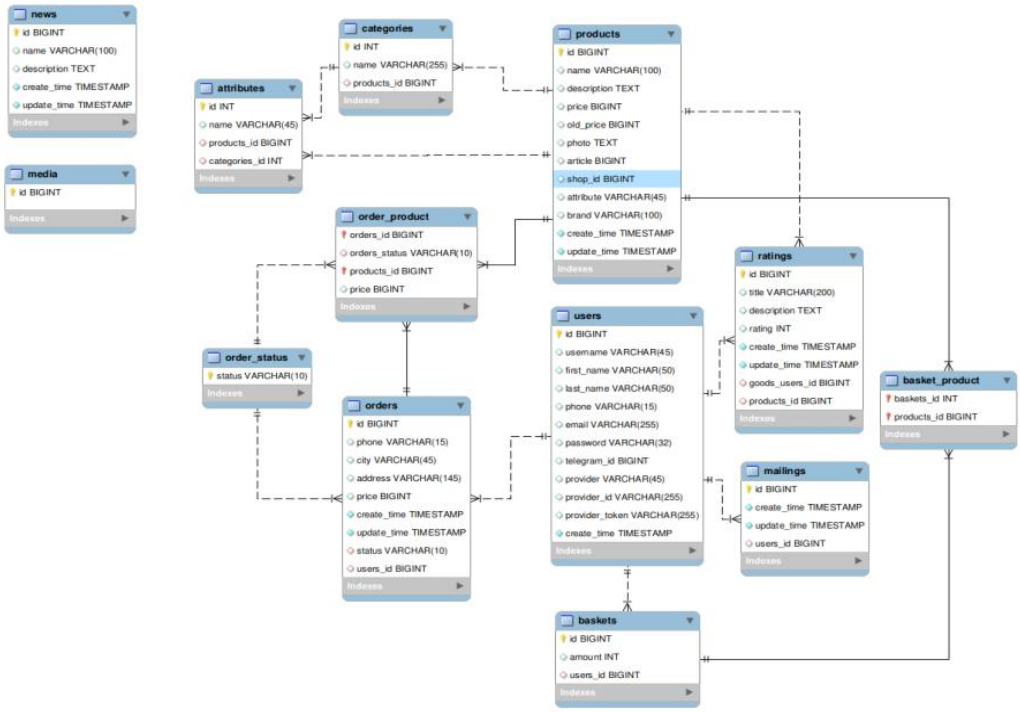

Fig. 2. Database schema

```
sudo apt-get update
sudo apt-get install curl
sudo apt-get install php php-curl
curl -sS https://getcomposer.org/installer -o composer-setup.php
sudo php composer-setup.php --install-dir=/usr/local/bin
--filename=composer
sudo composer self-update
composer create-project laravel/laravel shop-fiscal
```

Fig. 3. Creating a new Laravel project

As an example of describing a relationship in Laravel, we can take the entities "User" and "Order". This will be a "one-to-many" relationship, which means that one entity can have a relationship with many other models of another entity. In our case, one user can have many orders, but one order belongs to only one user. In the framework, this is described as a user has many orders ("hasMany"), and an order belongs to a user ("belongsTo"). An example of this relationship is shown in Fig. 4.

To implement routes and define all URLs, they need to be described in the web.php file. However, since we are developing not just a regular website but a large e-commerce platform using Blade and REST API. The routing logic is organized across five route files:

- "web.php" – the main file where routes for the client-side, built with the Blade templating engine, are described and where the other four files are included;
- "web-auth.php" – a file that defines routes for user authentication;
- "api.php" – a file where routes for the client-side, built with REST API, are defined;
- "webhook.php" – a file that contains routes for webhooks, such as payment system callbacks;
- "admin.php" – a file where routes for the admin panel, including all CRUD operations, are defined.

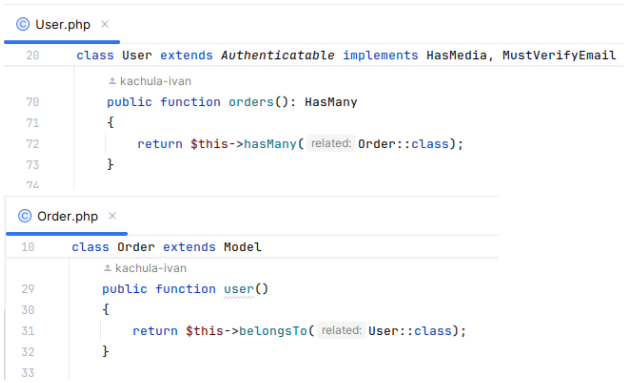

Fig. 4. User connection to orders

In these route files, not only the URLs for accessing specific content are specified, but also middleware is assigned. The middleware includes:

- "Guest" – allows access to a route only for non-authenticated users;
- "Auth" – the opposite of Guest, allowing access only to authenticated users;
- "Admin" – a custom middleware that grants access only to site managers.

Now, the next step is to implement request handling and business logic. One of the most crucial business logic components in e-commerce is the shopping cart functionality. It involves numerous validations and must always remain in an active state.

For a user to see the website, the request follows a structured path: first, the user accesses a URL that corresponds to a specific route. This route directs the request to a controller, where the required data is gathered and processed. The interaction with incoming data happens within the controller.

- In REST API development, the controller returns JSON data, which is then parsed on the frontend.
- In Blade templating, the controller prepares data and sends it to a View, where the layout and final rendering take place using predefined variables.

Fig. 5 presents the finalized homepage of the website.

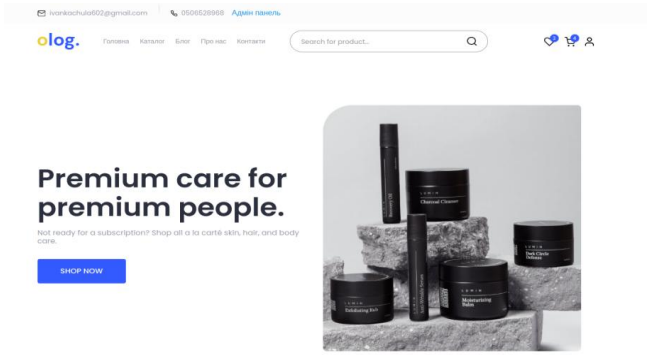

Fig. 5. Home page of the site

The system architecture adheres to the "thin controllers, fat models" principle by delegating business logic to dedicated service classes and facade components rather than implementing it directly within controllers. These classes will handle the core implementation, while controllers will simply call the necessary functionality.

To optimize the shopping cart and reduce database load, the cart identifier will be stored in the browser's cookies. This identifier will be retrieved from the local environment without being stored on the server. Using this identifier, we can efficiently fetch cart data and the associated products from the database.

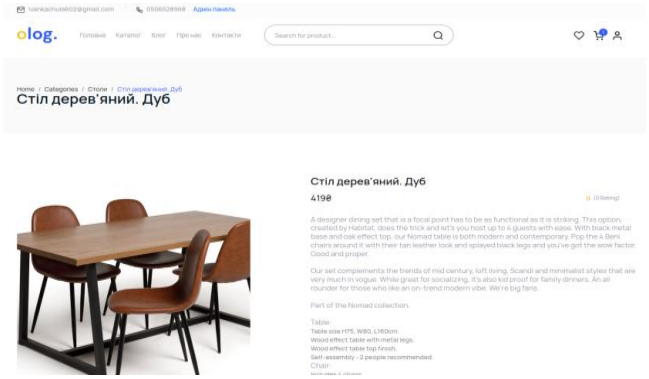

Fig. 6. Page of the product added to the cart

Fig. 6 illustrates the appearance of a product that has been added to the cart.

The administrative section of the website includes all the essential functions needed for an online store, from adding new products to processing orders directly from the admin panel. The admin panel is designed to be simple and lightweight to avoid unnecessary system load and ensure fast performance. This also allows users to quickly locate all necessary functions.

Usability aspects of the platform were evaluated through scenario-based testing involving representative business users performing typical administrative tasks, such as order processing, receipt generation, and report retrieval. The evaluation focused on ease of navigation, clarity of interface elements, and task completion efficiency.

The results indicate that the system supports intuitive interaction and requires minimal training for effective use. The centralized administrative panel and clear separation of functional components contribute to positive user perception and facilitate system adoption in real-world e-commerce environments.

The entire menu is conveniently placed on the left sidebar, making navigation intuitive and straightforward. Additionally, administrators can access logs and use the Tinker console without directly logging into the server where the website repository is hosted.

All dynamic site elements, such as phone numbers, emails, contact information, and copyright details, are grouped separately. This allows for dynamic updates to the footer content without modifying the HTML structure. Such an approach greatly simplifies the process of changing static data – users only need to access the admin panel instead of manually editing code on the remote server.

Fig. 7 illustrates the appearance of the product management page within the administrative section of the website.

To implement the main functionality of receipt fiscalisation, we applied an approach similar to the cart system. A façade and controller were created to handle incoming data. However, since this is the most complex functionality, this setup alone was not sufficient. Therefore, several additional "Laravel actions" were created, organizing the implementation of different functions across various files, such as receipt creation or report generation.

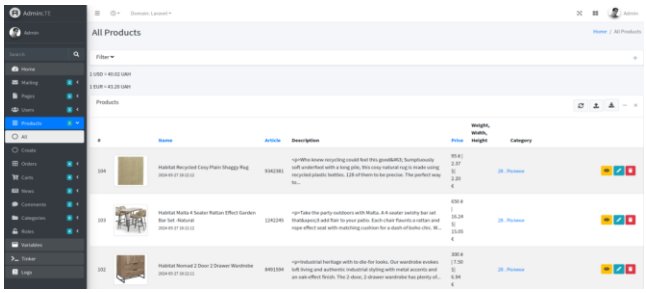

Fig. 7. All products page on the administrative part of the site

Here's how the entire fiscalisation process works from the inside, considering the code structure approach:

1. The user accesses the page through a route.
2. The controller receives all the sent data and calls the facade.
3. The facade establishes a connection with the Checkbox API and executes the "laravel action."
4. The action performs a specific request to the already open Checkbox API connection and returns the result.
5. All the data is returned to the controller, which processes it and responds with the receipt.

This structured approach ensures smooth and efficient fiscalisation and receipt processing within the system

The REST API integration with the Checkbox service follows best practices for secure and reliable communication. Authentication tokens are stored securely in environment configuration files and automatically refreshed when expired. This approach prevents unauthorized access and reduces manual intervention.

To comply with API rate limiting constraints, request throttling mechanisms are applied, ensuring that the number of requests remains within the allowed limits defined by the service provider. Timeout thresholds are configured to prevent long-running requests from blocking system processes.

In the event of API failures, the system distinguishes between temporary and critical errors. Temporary issues trigger retries, while persistent failures result in fallback responses and administrative alerts. This design ensures uninterrupted platform operation and consistent fiscal reporting even under unstable external API conditions.

In Fig. 8 you can see the implementation of the check fiscalisation functionality itself in Laravel action.

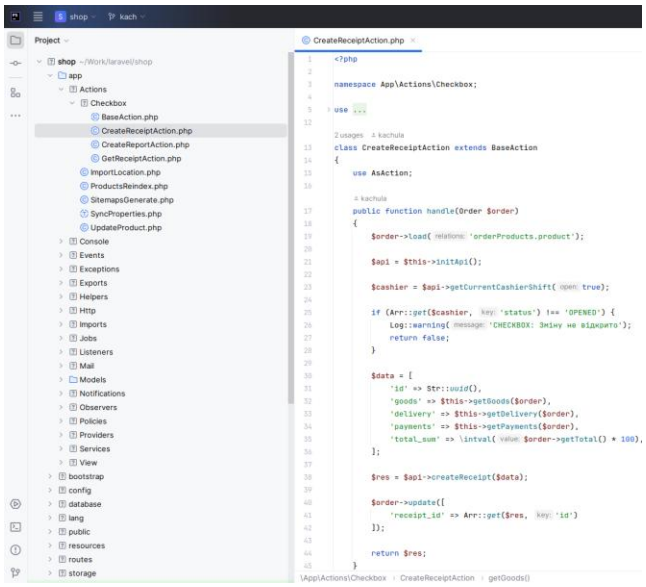

Fig. 8. Implementation of check fiscalisation

Security was a critical aspect of the proposed platform architecture due to the processing of fiscal and personal data. Secure API communication with the Checkbox service is ensured through HTTPS and token-based authentication mechanisms. All authentication credentials and API keys are stored in protected environment configuration files and are not hardcoded within the application logic.

Role-based access control is implemented using middleware to restrict access to administrative and fiscalisation functions. Additionally, Laravel's built-in protection mechanisms, including request validation and CSRF protection, are utilized to prevent unauthorized actions. From a data protection perspective, the system follows the principle of minimal data storage, retaining only information required for fiscal reporting and business operations. This approach supports compliance with applicable data protection regulations and reduces risks associated with unauthorized data access.

In Fig. 8 you can see that we first load all the data, this is done to optimize the process of interaction with the database. Then we initiate a connection to the Checkbox API through the facade and through it we check whether the working shift is open. If the shift is not open, then we return with an error where we indicate this, because fiscalisation is impossible with a closed shift. If the shift is open, then an array with data is formed and sent to the facade, which will already add all the authorization tokens to the request header and send the prepared data to Checkbox where the check is already fiscalised and returned.

In addition to basic validation checks demonstrated in Fig. 8, the fiscalisation module incorporates a multi-level error handling strategy. At the application level, request validation ensures data completeness and correctness before API calls are made. At the integration level, all interactions with the Checkbox API are wrapped in exception handling mechanisms to capture network errors, invalid responses, and authorization failures. To enhance system reliability, retry mechanisms are implemented for transient failures such as network timeouts, while critical errors (e.g., closed shifts or invalid credentials) trigger immediate user notifications and logging. All fiscalisation-related errors are recorded in system logs for audit and debugging purposes, ensuring traceability and compliance with fiscal regulations. This approach minimizes transaction loss risks and ensures stable system behavior under real-world operating conditions.

You can obtain the receipt by clicking on the yellow button with a receipt icon. The form is located in the administrative section of the website, on the page with all orders [12].

According to Article 10 of the Constitution of Ukraine, the state language of Ukraine is Ukrainian. The Law of Ukraine "On Ensuring the Functioning of the Ukrainian Language as the State Language" regulates the use of Ukrainian Language in public spheres. This implies that all official documents, including cash receipts, must be issued in Ukrainian.

However, when of software-based cash register systems or software-based of software-based cash register systems, a foreign language may be used alongside Ukrainian, provided it duplicates the information presented in Ukrainian. The absence of mandatory details in the state language would result in a violation of Ukrainian legislation [7].

Thus, cash receipts in Ukraine must be issued in Ukrainian, though duplication in another language is permissible, provided all mandatory information is present in Ukrainian.

At the very top of the check is the name, location and ID of the organization. A little lower is the list of goods. One item consists of a barcode of the product, name, number of pieces, price per piece, total price, discount, final price of the item. Next comes the payment data, namely the type of payment, if it is a card payment, then the card mask, payment system. Below is the total amount of the order. And at the very end is information about the check itself, namely the check number, date of its creation, type of creation online or offline, fiscal number of the of software-based cash register systems and on the right side a QR code to the taxpayer's electronic account, in which you can check the check already in the state body. In Fig. 9 you can see what a check that is already returned from Checkbox looks like.

To generate any of the reports, in the administrative section, you need to click on the corresponding button for the report on the order list page. X-reports can be generated at any time, as long as the register is open, and their creation is mainly for informational purposes. The X-report contains a summary of the receipts created within a specific period. It includes the receipt ID, its type (sale or return), date and time, fiscal number, payment type, amount in UAH, amount including rounding, local PPR register number, receipt type (online or offline), order discount, and a link to the receipt.

Fig. 9. Ready-made check for order

On the other hand, a Z-report can only be generated once at the end of a shift because these data will be sent to the tax authorities. This report is more detailed and contains general information. While each line of the X-report represents an individual receipt with its detailed information, each line in the Z-report represents a shift, and the report presents overall information for all cash registers, including their receipts. The Z-report contains the following data: register address, shift opening/closing date, total number of receipts, fiscal number of the register, shift number, revenue, date and time of the last fiscal receipt, number of receipts for purchases and returns, VAT amount, and tax totals. All of this information is mandatory because the tax authorities monitor these processes.

For testing the fiscalisation and reporting functionality, the Checkbox service provides test API keys, a test cash register, and a cashier. However, the process is identical to real cash registers and cashiers. All stages follow the exact sequence as in real fiscalisation. Fig. 10 shows the test data provided by the Checkbox service.

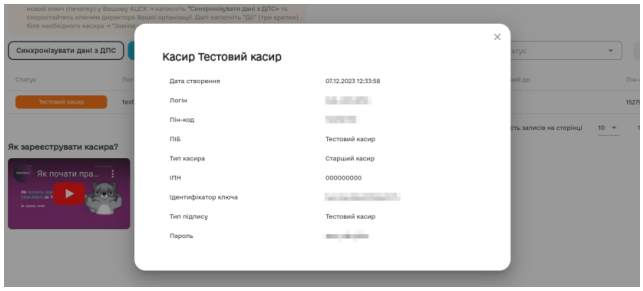

Fig. 10. Test cashier data provided by Checkbox service

The main data required to open a shift and start operations are the license key for the cash register and the cashier's login, password, and PIN code. These data are confidential and must not be shared with anyone [9, 10].

To begin working in test mode, all that is needed is to fill in the test keys in the ".env" file. Once this is done, everything is set up and should work as expected. The first step is to open the shift, after which all necessary fiscalisation or reporting operations can be performed. All main functions will operate as they would in real-life situations.

For test receipts, there are also settings for their appearance and other configurations. You can not only set their graphic appearance but also the appearance of the receipt in a text file. However, visually, it differs from a regular receipt by a large red warning stating that this is a test receipt, and this warning cannot be removed or modified. This text will disappear only for fully fiscalised receipts.

A test receipt cannot be used during actual operations, as it is not considered fiscal and therefore cannot be sent to customers. A test receipt does not go to the tax authorities and does not undergo fiscalisation through them. However, if a test receipt is created and sent, everything will work as expected for real ones. The test receipt is solely for debugging its appearance and the overall functioning of the online store through all stages of fiscalisation.

Test reports can also be created for both X-reports and Z-reports. They are generated for test tokens and are fully identical to how real reports would be generated. Fig. 11 shows an example of how an X-report arrives via email as a message with an attached file.

Fig. 11. Letter with X-report

The Checkbox API service provides powerful capabilities for testing and debugging the process from within. Almost all the functions available in a real working cash register can be tested on the test one. For testing, there is no need to sign contracts, verify documents, or anything similar. All that is required is to register on the service and enter the test tokens, which are provided immediately after registration [6, 11].

To assess the operational performance of the proposed fiscalisation architecture, a series of experimental tests were conducted in the Checkbox test environment. The evaluation focused on API response time, system stability under load, and consistency of fiscalisation operations.

Response time measurements were performed for receipt creation requests under normal operating conditions. The average API response time ranged from 350 ms to 420 ms, with peak values not exceeding 500 ms. These results indicate that the fiscalisation process does not introduce significant delays in the checkout workflow.

Load testing was conducted by simulating up to 100 consecutive fiscalisation requests within a single session. The system maintained stable operation without data loss or incorrect fiscal receipts. No critical failures were observed, although response times increased by approximately 15% under peak load conditions.

Despite the advantages of the proposed solution, several limitations and risks should be considered. First, the system is dependent on third-party fiscalisation services, making it vulnerable to API changes, service outages, or rate-limiting constraints. Second, real-time fiscalisation requires stable network connectivity; disruptions may affect transaction processing and require offline handling strategies. Regulatory risks also exist, as fiscal legislation may change, necessitating updates to system logic, data structures, and reporting formats. From a scalability perspective, while the proposed architecture is suitable for small and medium-sized e-commerce platforms, large-scale deployments may require additional performance optimization, asynchronous processing, and load-balancing mechanisms. These limitations define important directions for future research, including cross-platform benchmarking, fault-tolerant fiscalisation mechanisms, and adaptive architectural models for dynamic regulatory environments.

4. Conclusions

This paper presented the design and implementation of a web-based e-commerce platform integrated with a software-based fiscal cash register system using a REST API. The proposed solution enables automated receipt fiscalisation, generation of X and Z fiscal reports, and direct interaction with tax authorities in compliance with current Ukrainian tax regulations. While the implementation is demonstrated using Ukrainian fiscal regulations, the proposed system architecture and integration approach are applicable to a broader range of digital fiscalisation frameworks and may serve as a reference model for other regulatory environments.

The conducted empirical evaluation demonstrated that software-based fiscalisation can be effectively integrated into real-time e-commerce workflows while maintaining acceptable system performance and operational reliability. The developed architecture supports automated fiscal reporting, reduces manual intervention, and ensures consistent processing of fiscal transactions for online sales scenarios.

Identified limitations related to external service dependency, network availability, and regulatory changes define directions for future research. Further work may focus on performance optimization, scalability improvements for large-scale deployments, and the development of offline handling mechanisms to enhance system robustness.

Future research may extend this work by conducting systematic load testing, response time analysis, and scalability evaluation under high transaction volumes.

References

[1] Andrushchenko, V. L., & Tuchak, T. V. (2024). Fiscalization: EU versus Ukraine. *Business Inform*, *5*(556), 264–270. https://doi.org/10.32983/2222-4459-2024-5-264-270

[2] Denda, S., Petrović, M. D., Vuksanović-Macura, Z., Radovanović, M. M., & Ely-Ledesma, E. (2024). What Are the Current Directions in the Local Marketplaces Fiscalization? The Online Media Content Analysis. *Societies*, *14*(4), 53. https://doi.org/10.3390/soc14040053

[3] Dimura, M. (2024, August 30). E-commerce in Ukraine: Figures, facts, prospects of online commerce development. *Business Site Agency Blog*. https://www.site2b.ua/en/web-blog-en/e-commerce-in-ukraine-figures-facts-prospects-of-online-commerce-development.html

[4] Hesami, S., Jenkins, H., & Jenkins, G. P. (2024). Digital Transformation of Tax Administration and Compliance: A Systematic Literature Review on E-Invoicing and Prefilled Returns. *Digital Government: Research and Practice*, *5*(3), 1–20. https://doi.org/10.1145/3643687

[5] Samchuk, L., & Povstiana, Y. (2024). UML diagrams of the management system of maintenance stations. *Informatyka, Automatyka, Pomiary w Gospodarce i Ochronie Środowiska*, *14*(4), 141–145. https://doi.org/10.35784/iapgos.6320

[6] Shepel, I. V. (2024). Maintaining Accounting Forms and Maintaining Accounting and Cash Flows in Transactional Operations. *BTU Kharkiv Repository*. https://repo.btu.kharkov.ua/bitstream/123456789/44269/1/tezy-dop-conf-02-11-23-86-87.pdf

[7] Verkhovna Rada of Ukraine. (2019). *Law of Ukraine on Ensuring the Functioning of the Ukrainian Language as the State Language.* https://ips.ligazakon.net/document/T192704?an=1

[8] Yang, C., Yang, L., Song, Y., & Fu, X. (2023). Paper Invoice Dilemma and Electronic Invoice Solution. *Frontiers in Business, Economics and Management*, *12*(1), 12–21. https://doi.org/10.54097/fbem.v12i1.13620

[9] Accounting outsourcing services (2024, April 19). *Current Trends in the Registration of Cash Registers and Payment Terminals in 2024: An Expert's View*. https://xn----7sbcickxnhd6alkoeinny1mti.xn--j1amh/uk/zrostayuchij-popit-na-programni-rro-vid-velikix-platnikiv-podatkiv

[10] Checkbox (2024, April 21). https://checkbox.ua

[11] Checkbox (2024). https://wiki.checkbox.ua/uk/portal/settings/receipts

[12] Exellio (2020, October 15). https://rro-info.com.ua/osoblyvosti-fiskalnogo-cheku

Ph.D Yuliia Povstiana
e-mail: yuliapovstyana@ukr.net

Candidate of Technical Sciences, associate professor, Faculty of Computer and Information Technologies, Department of Software Engineering, Lutsk National Technical University.
Research interest: soft skills, digital and communication technologies in education, computer graphics and image processing, and analysis using UML diagrams. Author of more than 80 scientific publications.

https://orcid.org/0000-0001-5426-4157

Ph.D. Lyudmila Samchuk
e-mail: samchuk204@gmail.com

Candidate of Technical Sciences, associate professor, Faculty of Transport and Mechanical Engineering, Department of Applied Mechanics and Mechatronics, Lutsk National Technical University.
Research interest: technology for manufacturing porous permeable materials in SHS mode; utilization of grinding sludge from bearing production; software tools for visualizing the results of SHS experiments and structural parameters of materials; interactive visualization of data from research on porosity and permeability of materials.

https://orcid.org/0000-0003-2516-045X

B.Sc. Ivan Kachula
e-mail: ka4ivan.dev@gmail.com

Student at the master's academic level, Lutsk National Technical University.
Research interest: web application development and design, Al, testing.

https://orcid.org/0009-0002-8483-9315