ДОДАТОК А

Апробація наукових результатів дослідження

Міністерство освіти і науки України

NURE

Харківський національний університет радіоелектроніки

ЗБІРНИК

студентських наукових статей

«Автоматизація та приладобудування»

«Automation and Development of Electronic Devices»

ADED-2024

(Випуск 2)

[електронне видання]

KITAP

http://nure.ua/department/kafedra-komp-yuterno-integrovanih-tehnologiy-avtomatizatsiyi-ta-mehatroniki-kitam

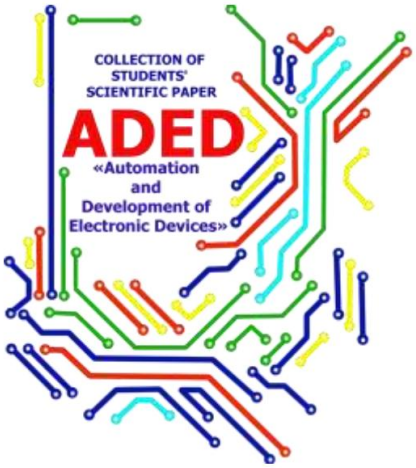
iTeD

http://itez.zntu.edu.ua/

http://kafea.kdu.edu.ua

Харків 2024

Міністерство освіти і науки України
Харківський національний університет радіоелектроніки
кафедра комп'ютерно-інтегрованих технологій, автоматизації та робототехніки
(КІТАР)



ЗБІРНИК

студентських наукових статей

«Автоматизація та приладобудування»

«Automation and Development of Electronic Devices»

ADED-2024

(Випуск 2)

[електронне видання]

Харків 2024

Автоматизація та Приладобудування («Automation and Development of Electronic Devices» ADED-2024) [Електронний ресурс] : збірник студентських наукових статей / Харківський національний університет радіоелектроніки ; [редкол.: І.Ш. Невлюдов та ін.]. – Харків : ХНУРЕ, 2024. – Вип. 2. – 290с.

Collection of Students' Scientific Paper «Automation and Development Of Electronic Devices» ADED-2024 Part 2 (Key infrastructure 2024) - Kharkiv/ The Editorial.: Nevlyudov I.Sh. (head), that all. Kharkiv: Kind of Kharkiv National Universitety of Radio Elektronics [electronic edition], 2024. – 290p with.

Рекомендовано рішенням
Науково-технічної ради
Харківського національного
університету радіоелектроніки
протокол №6 від 29.11.2018

Рекомендовано рішенням Вченої ради
факультету Автоматики і комп'ютеризованих технологій
Харківського національного
університету радіоелектроніки
протокол № 4 від 26.12.2024

Збірник містить наукові статті здобувачів першого (бакалаврського), другого (магістерського) рівнів вищої освіти кафедри комп'ютерно-інтегрованих технологій, автоматизації та робототехніки (КІТАР) Харківського національного університету радіоелектроніки, кафедри Інформаційних технологій електронних засобів (ІТЕД) Запорізького національного технічного університету та кафедри Електронних апаратів (ЕА) Кременчуцького національного університету ім. М. Остроградського які навчаються за спеціальностями: 151 Автоматизація та комп'ютерно-інтегровані технології, 174 Автоматизація, комп'ютерно-інтегровані технології та робототехніка; 172 Телекомунікації та радіотехніка, 171 Електроніка та 163 Біомедична інженерія. Статті надані в авторській редакції.

УДК 004.658

SRM-СИСТЕМА В СУЧАСНОМУ ПІДПРИЄМСТВІ: ЕФЕКТИВНЕ УПРАВЛІННЯ БІЗНЕС-ПРОЦЕСАМИ

Я.В. Олінкевич
Харківський національний університет радіоелектроніки
Україна, 61166, Харків, пр. Науки 14
E-mail: yaroslav.olinkevych@nure.ua

Анотація: У даній статті досліджено роль системи SRM у промисловому виробництві. Розглянуто основні функціональні можливості, етапи та особливості впровадження кастомізованої SRM, визначено типові помилки при виборі готового програмного рішення, проаналізовано переваги та недоліки впровадження системи.

Ключові слова: SRM, постачальник, взаємовідносини, підприємство, виробництво.

SRM SYSTEM IN A MODERN ENTERPRISE: EFFECTIVE MANAGEMENT OF BUSINESS PROCESSES

Y. Olinkevych
Kharkiv Kharkiv National University of Radio Electronics
Ukraine, 61166, Kharkiv, Nauky av, 14
E-mail: yaroslav.olinkevych@nure.ua

Abstract: This article examines the role of the SRM system in industrial production. The main functionalities, stages and peculiarities of implementation of a customised SRM are considered, typical mistakes when choosing a ready-made software solution are identified, advantages and disadvantages of implementing the system are analysed.

Key words: SRM, supplier, relationship, enterprise, production.

У сучасному бізнес-середовищі, що характеризується швидкою зміною умов та високою конкуренцією, критично важливим є ефективне управління виробничими ресурсами. Одним із провідних аспектів даного елементу господарської діяльності є робота з постачальниками, грамотна організація взаємовідносин із якими здатна допомогти значно оптимізувати витрати на підтримку бізнес-процесів, досягти вигідних комерційних умов співпраці, а також мінімізувати ризики. Впровадження системи SRM дозволяє спростити та автоматизувати ці процеси, створюючи єдину платформу для управління партнерськими зв'язками (рис. 1)

Рисунок 1 – Система управління взаємовідносинами з постачальниками

SRM (Supplier Relationship Management) – це програмне забезпечення (ПЗ), створене для керування відносинами з постачальниками в корпоративному середовищі підприємства. Таке ПЗ дозволяє компаніям обирати серед постачальників товарів і послуг тих партнерів, що найкраще відповідають їхнім очікуванням, а кінцевою метою використання SRM є максимізація доходів підприємства при одночасному зниженні його витрат [1]. Основні функції SRM-системи можна розділити на кілька ключових категорій:

а) управління даними постачальників. Централізоване зберігання даних дозволяє мати повний доступ до контактних даних, договорів та історії співпраці з постачальниками. Завдяки сегментації постачальників, підприємство може класифікувати їх за ефективністю, стратегічною важливістю тощо. Також SRM-система забезпечує можливість оцінювати можливі ризики, аналізуючи фінансовий стан, репутацію та відповідність вимогам сертифікації;

б) планування та комунікація. SRM-система сприяє ефективному плануванню закупівель. Також завдяки інтеграції з ERP-системою досягається автоматизація процесів закупівель, що дозволяє уникати дублювання дій та інших помилок, причиною яких є людський фактор. Інструменти для комунікації забезпечують швидкий і зручний зв'язок, обмін документами та запитами між постачальниками й підприємством;

в) управління процесами закупівель. Ефективність процесів закупівель значно покращується через створення тендерів, порівняння наявних на ринку пропозицій та вибір найкращих умов для ділової угоди. SRM-системи дозволяють контролювати всі етапи виконання договорів завдяки зручному моніторингу строків постачання, обсягів продукції тощо. Також інтеграція з фінансовими інструментами дозволяє більш ефективно здійснювати облік рахунків, проводити розрахунки та узгоджувати фінансові операції;

г) аналіз та оцінка постачальників. SRM-система дозволяє проводити оцінку ефективності та надійності постачальників шляхом аналізу ключових показників, таких як своєчасність доставок, якість товарів, послуг тощо. Рейтинг постачальників допомагає підприємству ухвалювати більш обґрунтовані рішення під час вибору потенційних партнерів. Звіти та аналітичні дані надають деталізовану інформацію про витрати, продуктивність співпраці та можливі ризики;

д) поліпшення якості взаємовідносин. Через прозорі механізми взаємодії компанії та постачальників вдається швидко вирішувати конфлікти та уникати непорозумінь. Завдяки розумній організації багатьох бізнес-процесів обидві сторони отримують рівноцінну економічну вигоду: зменшуються витрати на забезпечення логістики, скорочується час обробки та виконання замовлень[2];

е) підтримка комплаєнсу та ризик-менеджмент. Система забезпечує контроль юридичної відповідності умов договорів чинним нормам і законодавству. Це дозволяє знизити ризики, пов'язані з несвоєчасними постачаннями чи фінансовими проблемами постачальників, завдяки своєчасному попередженню про потенційні загрози.

Модулі та утиліти, що будуть доступними для використання в SRM-системі, є опціональними для кожного окремого виробництва, однак існує перелік типових інструментів, які найчастіше зустрічаються в складі програмного пакету [3]. Наприклад, модуль проєктування фокусується на підтримці процесу розробки продукції, де інтеграція з постачальниками дозволяє отримувати критично важливі дані, такі як специфікації матеріалів, параметри постачання та інновації у виробництві. Це допомагає підвищити якість кінцевого продукту, зменшити витрати на розробку та скоротити час виходу продукту на ринок. Основною задачею модулю пошуку постачальників є оптимізація процесу вибору постачальників і товарів. Він забезпечує аналіз ринку, моніторинг пропозицій, оцінку потенційних постачальників та їх відповідності потребам компанії. Завдяки цьому

підприємство може вибирати найбільш конкурентоспроможні товари та послуги, мінімізуючи ризики. Модуль із закупівель підтримує управління процесами закупівлі, починаючи від формування потреб і планування, закінчуючи оформленням замовлень. Він дозволяє автоматизувати створення заявок, відстежувати виконання постачання та аналізувати витрати, що сприяє ефективному управлінню ресурсами підприємства. Модуль управління в основному стосується інвентаризації, контролю бази постачальників і видимість ланцюгів постачання та, як і у випадку з іншим модулем із виробництва, що допомагає з управлінням якістю та деякими інженерними змінами, здебільшого є корисним доповненням до основного функціоналу ERP-системи. Модуль врегулювання автоматизує фінансові операції між підприємством і постачальниками. Завдяки інтеграції з іншими модулями, модуль дозволяє уникнути розбіжностей між замовленнями, накладними та фактично поставленими товарами. Також важливим є наявність інструментів із підтримки операційного планування, управління проєктами, контролем бюджетів та створення специфікацій

Впровадження SRM-системи є комплексним процесом, який вимагає ретельного планування. Вибір оптимальної системи передбачає порівняльний аналіз наявних рішень на ринку та оцінку їхньої відповідності корпоративним потребам. Якщо стандартні рішення не повністю задовольняють вимоги підприємства, доцільним може бути розроблення кастомізованої SRM-системи. Етапи впровадження кастомізованої SRM-системи включають послідовну реалізацію взаємопов'язаних кроків [4]:

1) Проведення аналітики бізнесу

Необхідно провести глибокий аналіз поточних бізнес-процесів управління взаємовідносинами з постачальниками, визначити больові точки та очікувані результати від впровадження системи.

2) Розробка прототипу

Здійснюється формулювання детальних вимог, розробка технічного завдання та плану впровадження, враховуючи специфіку конкретного підприємства. Особливо важливим моментом є ситуація, коли на підприємстві вже впроваджена ERP-система. У такому випадку SRM-рішення має бути максимально інтегрованою з наявною інформаційною інфраструктурою. Необхідно забезпечити безшовну взаємодію між системами, уникнути дублювання даних та створити уніфіковану платформу для ефективного управління бізнес-діяльністю підприємства.

3) Створення програмного забезпечення

Розробка ПЗ відбувається з урахуванням вимог конкретного підприємства. Створюється інтуїтивно зрозумілий інтерфейс, що наповнюється необхідним функціоналом, максимально адаптованим до бізнес-процесів підприємства. Важливим аспектом даного етапу є всебічне тестування, яке дозволяє виявити та усунути потенційні недоліки ще до впровадження продукту.

4) Впровадження системи в підприємство

Цей етап передбачає поступове впровадження SRM-системи в роботу підприємства з інтеграцією з наявними бізнес-системами. Одночасно проводиться налаштування системи відповідно до конкретних вимог виробництва та підготовка до роботи з реальними даними. Спочатку система тестується на обмеженій групі користувачів або на окремих підрозділах компанії. Після успішного тестування відбувається масштабування та інтеграція системи на рівні всієї організації. Важливою складовою є навчання співробітників, які будуть працювати з системою.

5) Надання подальшої технічної підтримки

Після впровадження системи забезпечується її постійна підтримка для гарантування стабільної роботи. Це включає регулярне оновлення програмного забезпечення, виправлення можливих помилок та адаптацію системи до нових вимог бізнесу.

Якщо вибір пав на придбання готового програмного рішення, слід враховувати типові помилки, що допускаються під час вибору готового пакету SRM-системи. Часто компанії обирають SRM-систему без глибокого аналізу власних потреб та бізнес-процесів. Відсутність чітких вимог до функціоналу системи призводить до вибору рішення, яке не відповідає реальним завданням організації. Наприклад, система може мати надмірну кількість функцій, які не використовуватимуться, або ж бракувати ключових інструментів. Також слід обережно підходити до вибору універсальної системи, яка не адаптується до конкретних галузевих стандартів, що в деяких випадках може тягнути за собою значне зниження її ефективності та невідповідність початковим очікуванням, оскільки кожна галузь має власну специфіку роботи з постачальниками. Нехтування інтегрованістю SRM-системи може привести до розриву в обміні даними між підрозділами, що ускладнює прийняття оперативних рішень. Ще одним надважливим аспектом є неврахування перспективи масштабування, коли компанії обирають програмний пакет, орієнтуючись на поточний обсяг діяльності, і не беруть до уваги можливість масштабування, що створює проблеми у разі розширення бізнесу чи збільшення кількості постачальників.

Впровадження SRM-системи має ряд суттєвих переваг для підприємства:

- оптимізація витрат. Автоматизація процесів закупівель, можливість порівняння цін та умов різних постачальників, а також ефективне планування призводять до значного скорочення операційних витрат;
- підвищення прозорості. Централізоване зберігання даних та автоматизація процесів забезпечують повну видимість усіх операцій з постачальниками, що знижує ризики шахрайства та помилок;
- покращення якості співпраці. Ефективні комунікаційні інструменти та чіткі процеси взаємодії сприяють побудові міцніших партнерських відносин з постачальниками;
- поліпшення аналітики. Доступ до детальної аналітичної інформації допомагає приймати більш обгрунтовані рішення щодо вибору постачальників та оптимізації закупівель.

Проте існують і певні недоліки та ризики впровадження SRM-системи:

- високі початкові витрати. Впровадження системи потребує значних інвестицій у програмне забезпечення, навчання персоналу та можливу модернізацію IT-інфраструктури;
- складність впровадження. Процес впровадження може бути тривалим і складним, особливо для великих підприємств з розгалуженою структурою;
- опір персоналу. Співробітники можуть чинити опір змінам та новим методам роботи, що вимагає додаткових зусиль для навчання та адаптації;
- технічні ризики. Можливі технічні збої, проблеми під час інтеграції з існуючими системами та питання безпеки даних;
- залежність від постачальників. У випадку використання готового програмного рішення підприємство стає залежним від постачальника ПЗ щодо оновлень та технічної підтримки.

ВИСНОВКИ. Таким чином, впровадження SRM-системи є важливим кроком для ефективного управління бізнес-процесами сучасного підприємства. Таке програмне забезпечення дозволяє не лише оптимізувати витрати та підвищити ефективність роботи з постачальниками, але й створює міцну основу для подальшого розвитку та масштабування бізнесу. В умовах зростаючої конкуренції та необхідності постійної оптимізації бізнес-процесів, впровадження SRM-системи стає важливим чинником для успішного функціонування підприємства на сучасному ринку.

ЛІТЕРАТУРА

1. What is SRM and why is it important for your business in 2024?. Weproc. URL: https://blog.weproc.com/en/supplier-management/what-is-srm/#why-use-srm (дата звернення: 15.11.2024).

2. What is supplier relationship management? | Definition from TechTarget. Search ERP. URL: https://www.techtarget.com/searcherp/definition/supplier-relationship-management-SRM (дата звернення: 16.11.2024).

3. Best SRM Software [2024] - Select the Best SRM Software For You | TEC. Enterprise Software and ERP Selection Consulting Services - TEC | TEC. URL: https://www3.technologyevaluation.com/c/srm (дата звернення: 20.11.2024).

4. SRM-система: що це таке та як правильно впровадити SRM систему? - Wezom. URL: https://wezom.com.ua/ua/blog/srm-system (дата звернення: 22.11.2024).

Науковий керівник: *Хрустальова Софія Володимирівна, доцент кафедри КІТАР Харківського національного університету радіоелектроніки*

ДОДАТОК Б

Лістинг програми

Код основного файлу смарт-контракту ProcurementContract.sol:

```
pragma solidity ^0.8.0;
import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
/**
 * @title ProcurementContract
 * @notice Смарт-контракт для фіксації та автоматизації етапів специфікації закупівлі
 * в рамках SRM-підсистеми з повним життєвим циклом та інтегрованою оплатою.
 */
contract ProcurementContract {
    struct MaterialItem {
        string category;
        string materialName;
        uint256 quantity;
        uint256 unitCostWei;
    }
    enum State {
        Created,            // 0: Контракт створено та розгорнуто
        SignedByCompany,    // 1: Компанія підписала контракт
        SignedByProvider,   // 2: Постачальник підписав контракт
        InTransit,          // 3: Товар відправлено
        Delivered,          // 4: Постачальник відмітив доставку
        Fulfilled,          // 5: Компанія підтвердила виконання (відбувається оплата)
        Canceled           // 6: Контракт скасовано
    }
    State public currentState;
    address public companySigner;
    address public providerSigner;
    string public specificationId;
    string public providerEmail;
    uint256 public totalCostWei;
    uint256 public desiredDeliveryDateTimestamp;
    address public tokenAddress;
    MaterialItem[] public items;
    uint256 public companySignedAt;
    uint256 public providerSignedAt;
    uint256 public deliveredAt;
    uint256 public fulfilledAt;
    event StatusUpdate(address indexed initiator, State newState, uint256 timestamp);
    event ItemAdded(string materialName, uint256 quantity, uint256 unitCostWei);
    event PaymentExecuted(address indexed from, address indexed to, uint256 amount);
    modifier onlyCompanySigner() {
        require(msg.sender == companySigner, "PC: Only company signer is authorized.");
        _;
```

```
}
modifier onlyProviderSigner() {
    require(msg.sender == providerSigner, "PC: Only provider signer is authorized.");
    _;
}
modifier inState(State _state) {
    require(currentState == _state, "PC: Invalid contract state.");
    _;
}
/**
 * @notice Створення контракту менеджером закупівель.
 * @param _provider Адреса гаманця постачальника.
 * @param _specId ID специфікації з MongoDB.
 * @param _email E-mail постачальника.
 * @param _totalCost Загальна вартість у Wei.
 * @param _deliveryDate Мітка часу бажаної доставки.
 * @param _tokenAddress Адреса токена ERC-20 для оплати.
 */
constructor(
    address _provider,
    string memory _specId,
    string memory _email,
    uint256 _totalCost,
    uint256 _deliveryDate,
    address _tokenAddress
) {
    require(_provider != address(0), "PC: Invalid provider address.");
    require(_totalCost > 0, "PC: Total cost must be greater than zero.");
    require(_tokenAddress != address(0), "PC: Invalid token address.");
    companySigner = msg.sender;
    providerSigner = _provider;
    specificationId = _specId;
    providerEmail = _email;
    totalCostWei = _totalCost;
    desiredDeliveryDateTimestamp = _deliveryDate;
    tokenAddress = _tokenAddress;
    currentState = State.Created;
    emit StatusUpdate(msg.sender, State.Created, block.timestamp);
}
function addItem(
    string memory _category,
    string memory _materialName,
```

```
    uint256 _quantity,
    uint256 _unitCostWei
) public onlyCompanySigner inState(State.Created) {
    require(_quantity > 0, "PC: Quantity must be greater than zero.");
    items.push(MaterialItem({
        category: _category,
        materialName: _materialName,
        quantity: _quantity,
        unitCostWei: _unitCostWei
    }));
    emit ItemAdded(_materialName, _quantity, _unitCostWei);
}
function getItemsCount() public view returns (uint256) {
    return items.length;
}
function getItem(uint256 index) public view returns (
    string memory category,
    string memory materialName,
    uint256 quantity,
    uint256 unitCostWei
) {
    require(index < items.length, "PC: Item index out of bounds.");
    MaterialItem memory item = items[index];
    return (item.category, item.materialName, item.quantity, item.unitCostWei);
}
function signContractByCompany() public onlyCompanySigner inState(State.Created) {
    require(items.length > 0, "PC: Cannot sign contract without items.");
    currentState = State.SignedByCompany;
    companySignedAt = block.timestamp;
    emit StatusUpdate(msg.sender, State.SignedByCompany, block.timestamp);
}
function signContractByProvider() public onlyProviderSigner inState(State.SignedByCompany) {
    currentState = State.SignedByProvider;
    providerSignedAt = block.timestamp;
    emit StatusUpdate(msg.sender, State.SignedByProvider, block.timestamp);
}
function markInTransit() public onlyProviderSigner inState(State.SignedByProvider) {
    currentState = State.InTransit;
    emit StatusUpdate(msg.sender, State.InTransit, block.timestamp);
}
function markDelivered() public onlyProviderSigner {
    require(
```

```
        currentState == State.SignedByProvider || currentState == State.InTransit,
        "PC: Contract must be signed by both parties first."
    );
    currentState = State.Delivered;
    deliveredAt = block.timestamp;

    emit StatusUpdate(msg.sender, State.Delivered, block.timestamp);
}
/**
 * @notice Крок 5: Компанія підтверджує виконання контракту та ініціює оплату.
 * @dev Використовує transferFrom. Це вимагає, щоб companySigner попередньо надав
 * контракту дозвіл (allowance) на списання токенів.
 */
function confirmFulfillment() public onlyCompanySigner inState(State.Delivered) {
    IERC20 token = IERC20(tokenAddress);
    bool success = token.transferFrom(companySigner, providerSigner, totalCostWei);
    require(success, "PC: Token transfer failed. Check Company allowance or balance.");
    emit PaymentExecuted(companySigner, providerSigner, totalCostWei);
    currentState = State.Fulfilled;
    fulfilledAt = block.timestamp;
    emit StatusUpdate(msg.sender, State.Fulfilled, block.timestamp);
}
/**
 * @notice Скасування контракту (до доставки).
 */
function cancelContract() public {
    require(
        msg.sender == companySigner || msg.sender == providerSigner,
        "PC: Only company or provider can cancel."
    );
    require(
        currentState != State.Delivered &&
        currentState != State.Fulfilled,
        "PC: Cannot cancel after delivery or fulfillment."
    );
    currentState = State.Canceled;
    emit StatusUpdate(msg.sender, State.Canceled, block.timestamp);
}
 /**
 * @notice Отримати поточний статус контракту як рядок.
 */
function getStatusString() public view returns (string memory) {
```

```
            if (currentState == State.Created) return "Created";
            if (currentState == State.SignedByCompany) return "SignedByCompany";
            if (currentState == State.SignedByProvider) return "SignedByProvider";
            if (currentState == State.InTransit) return "InTransit";
            if (currentState == State.Delivered) return "Delivered";
            if (currentState == State.Fulfilled) return "Fulfilled";
            if (currentState == State.Canceled) return "Canceled";
            return "Unknown";
        }
        /**
         * @notice Перевірка, чи протермінована доставка.
         */
        function isOverdue() public view returns (bool) {
            return block.timestamp > desiredDeliveryDateTimestamp &&
                   currentState != State.Fulfilled &&
                   currentState != State.Canceled;
        }
    }
```

Код для створення тестового токену за стандартом ERC-20 TestToken.sol:

```
pragma solidity ^0.8.0;
import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
/**
 * @title TestToken
 * @notice Простий тестовий токен ERC-20.
 */
contract TestToken is ERC20 {
    constructor(uint256 initialSupply) ERC20("HNU Test Hryvnia", "THRN") {
        // Використав 10**18 для імітації стандартної одиниці (Ether/Wei)
        _mint(msg.sender, initialSupply * (10**18));
    }
}
```

```
import { ethers } from "ethers";
import dotenv from "dotenv";
import fs from "fs";
import path from "path";
import { fileURLToPath } from 'url';
dotenv.config();
const __filename = fileURLToPath(import.meta.url);
const __dirname = path.dirname(__filename);
const loadArtifact = (contractName) => {
      const artifactPath = path.join(__dirname, 'artifacts', 'contracts', `${contractName}.sol`,
`${contractName}.json`);
    return JSON.parse(fs.readFileSync(artifactPath, 'utf8'));
};
const deployAllContracts = async () => {
    try {
        const provider = new ethers.JsonRpcProvider("http://127.0.0.1:8545");
        const wallet = new ethers.Wallet(process.env.PRIVATE_KEY, provider);
        const deployerAddress = wallet.address;
        let nonce = await provider.getTransactionCount(wallet.address, 'pending');
        console.log(`\n=====================================================`);
        console.log(`Deployer Address: ${deployerAddress}`);
        console.log(`=====================================================\n`);
        const INITIAL_TOKEN_SUPPLY = 1_000_000;
        const tokenArtifact = loadArtifact('TestToken'); // Читаємо TestToken
         const TokenFactory = new ethers.ContractFactory(tokenArtifact.abi, tokenArtifact.bytecode,
wallet);
        console.log("1/2 Deploying TestToken...");
        const testToken = await TokenFactory.deploy(INITIAL_TOKEN_SUPPLY, { nonce: nonce++ });
        await testToken.waitForDeployment();
        const tokenAddress = await testToken.getAddress();
        console.log(`TestToken deployed to: ${tokenAddress}\n`);
        const procContractArtifact = loadArtifact('ProcurementContract');
              const ProcContractFactory = new ethers.ContractFactory(procContractArtifact.abi,
procContractArtifact.bytecode, wallet);
        const MOCK_SUPPLIER_ADDRESS = "0x70997970C51812dc3A010C7d01b50e0d17dc79C8".toLowerCase();
        const MOCK_TOTAL_COST_WEI = ethers.parseEther("1000.00");
        const MOCK_DELIVERY_DATE = Math.floor(Date.now() / 1000) + (24 * 3600);
        console.log("2/2 Deploying ProcurementContract...");
        const procurementContract = await ProcContractFactory.deploy(
            MOCK_SUPPLIER_ADDRESS,
            "MOCK-SPEC-ID",
            "test@supplier.com",
```

```
                MOCK_TOTAL_COST_WEI,
                MOCK_DELIVERY_DATE,
                tokenAddress,
                { nonce: nonce++ }
            );
            await procurementContract.waitForDeployment();
            const procContractAddress = await procurementContract.getAddress();
            console.log(`ProcurementContract deployed to: ${procContractAddress}\n`);
            const allowanceAmount = MOCK_TOTAL_COST_WEI;
            console.log(`Granting allowance (approve) to the ProcurementContract...`);
            const tokenContract = new ethers.Contract(tokenAddress, tokenArtifact.abi, wallet);
            const txApprove = await tokenContract.approve(
                procContractAddress,
                allowanceAmount,
                { nonce: nonce++ }
            );
            await txApprove.wait();
            console.log(`Allowance granted successfully!`);
        } catch (error) {
            console.error("FATAL Deployment chain error:", error);
            process.exit(1);
        }
    };
    deployAllContracts();
```

Приклад коду роутеру з логікою взаємодії серверу з блокчейн-модулем із файлу specifications.js:

```
router.post('/api/contract/:address/confirm-fulfillment',
auth, checkRole(['purchase_manager']), async (req, res) => {
  const contractAddress = req.params.address;
  const specId = req.body.specId;
  try {
    console.log('[FULFILL] Підтвердження виконання з оплатою...');
    const procurementArtifact = loadContractArtifact('ProcurementContract');
    const testTokenArtifact = loadContractArtifact('TestToken');
    const spec = await Specification.findById(specId);
    if (!spec) {
      return res.status(404).json({ error: 'Специфікацію не знайдено' });
    }
    const totalCostUAH = spec.materials.reduce((sum, item) => sum + (parseFloat(item.price) || 0), 0);
    const totalCostWei = ethers.parseEther(totalCostUAH.toFixed(2).toString());
    console.log('[FULFILL] Встановлення allowance для токена...');
    const tokenContract = new ethers.Contract(
      TEST_TOKEN_ADDRESS,
```

```
testTokenArtifact.abi,
    deployerWallet
  );
  let nonce = await provider.getTransactionCount(deployerWallet.address, 'pending');
  const approvalTx = await tokenContract.approve(
    contractAddress,
    totalCostWei,
    { nonce: nonce++ }
  );
  await approvalTx.wait();
  console.log('[FULFILL] Allowance встановлено');
  console.log('[FULFILL] Виклик confirmFulfillment()...');
  const contractInstance = new ethers.Contract(
    contractAddress,
    procurementArtifact.abi,
    deployerWallet
  );
  const tx = await contractInstance.confirmFulfillment({ nonce: nonce++ });
  console.log('[FULFILL] Транзакція відправлена:', tx.hash);
  const receipt = await tx.wait();
  console.log('[FULFILL] Транзакція підтверджена');
  await Specification.findByIdAndUpdate(specId, {
    blockchainContractStatus: 'Fulfilled',
    fullfilledHash: tx.hash,
    fulfilledMarkedDate: new Date(),
    paymentHash: tx.hash,
    isCompleted: true
  });
  console.log('[FULFILL] БД оновлена');
  res.json({
    success: true,
    message: 'Контракт виконано та оплачено.',
    txHash: tx.hash,
    amountPaid: `${totalCostUAH} UAH`
  });
} catch (error) {
  console.error('[FULFILL] ПОМИЛКА:', error);
  let errorMessage = error.message;
  if (error.message.includes('insufficient allowance')) {
    errorMessage = 'Недостатньо дозволу на переказ токенів. Спробуйте ще раз.';
  } else if (error.message.includes('insufficient balance')) {
    errorMessage = 'Недостатньо токенів на рахунку компанії для оплати.';
```

```
        } else if (error.message.includes('ERC20: transfer amount exceeds balance')) {
      errorMessage = 'На рахунку компанії недостатньо коштів.';
    }
    res.status(500).json({
      success: false,
      error: 'Не вдалося підтвердити виконання та оплатити.',
      message: errorMessage
    });
  }
});
```

Код моделі специфікації Specification.js:

```
const { Schema, model } = require('mongoose')
const materialSchema = new Schema({
  title: {
    type: String,
    required: true
  },
  quantity: {
    type: Number,
    required: true
  },
  price: {
    type: Number,
    required: true
  }
});
const specificationSchema = new Schema({
  date: {
    type: String,
    required: true
  },
  providerId: {
    type: Schema.Types.ObjectId,
    ref: 'Provider',
    required: true
  },
  blockchainAddress: {
    type: String,
    default: null
  },
  blockchainContractStatus: {
    type: String,
    enum: [
      'Created',
      'SignedByCompany',
      'SignedByProvider',
      'InTransit',
      'Delivered',
      'Fulfilled',
      'Canceled',
      'Disputed',
      'DeploymentError'
    ],
    default: 'Created'
  },
  createdBy: {
    name: String,
    id: String
  },
  signatureTransactionHash: {
    type: String,
    default: null
  },
  providerSignatureHash: {
    type: String,
    default: null
  },
  transitHash: {
```

```
    type: String,
    default: null
  },
  deliveredHash: {
    type: String,
    default: null
  },
  fullfilledHash: {
    type: String,
    default: null
  },
  canceledHash: {
    type: String,
    default: null
  },
  signedDate: {
    type: Date,
    default: null
  },
  providerSignedDate: {
    type: Date,
    default: null
  },
  transitMarkedDate: {
    type: Date,
    default: null
  },
  deliveredMarkedDate: {
    type: Date,
    default: null
  },
  fulfilledMarkedDate: {
    type: Date,
    default: null
  },
  canceledMarkedDate: {
    type: Date,
    default: null
  },
  canceledBy: {
    type: String,
    enum: ['Company', 'Provider'],
    default: null
  },
  provider: {
    type: String,
    required: true
  },
  email: {
    type: String,
    required: true
  },
  isPosted: {
    type: Boolean,
    required: true,
    default: false
  },
  postedDate: {
    type: Date
  },
  isDeleted: {
    type: Boolean,
    required: true,
    default: false
  },
  isCompleted: {
    type: Boolean,
    required: true,
    default: false
  },
  isOverdue: {
    type: Boolean,
    default: false
  },
  overdueDays: {
    type: Number,
    default: 0
  },
  comment: {
    type: String,
    default: null
  },
  materials: [materialSchema]
});
module.exports = model('Specification',
specificationSchema);
```

Код сторінки кабінету представника постачальника provider-cabinet.hbs:

```
<h2>Вітаємо, {{name}}! 👋</h2>
<div class="row">
    <div class="col s12 m6">
        <div class="card blue-grey darken-1">
            <div class="card-content white-text">
                <span class="card-title">Ваша Компанія: {{company.provider}}</span>
                <p><strong>E-mail:</strong> {{company.email}}</p>
                <p><strong>Контактний номер:</strong> {{company.phone}}</p>
                <hr>
                <h4>Ідентифікація для Блокчейну</h4>
                <p>
                    <strong>Публічна Адреса Гаманця:</strong>
                    <span style="font-family: monospace; font-size: 1.1em;">{{walletAddress}}</span>
                </p>
                <p class="small-text">
                        *Ця адреса є Вашим ідентифікатором у мережі Hardhat і використовується для
підписання всіх
                    смарт-контрактів.
                </p>
            </div>
        </div>
    </div>
</div>
<div class="row">
    <div class="col s12">
        <h3 class="provider-cabinet-table header">Специфікації, що потребують Вашої уваги</h3>
        <table class="provider-cabinet-table striped responsive-table">
            <thead>
                <tr>
                    <th>ID</th>
                    <th>Дата подання</th>
                    <th>Бажана дата доставки</th>
                    <th>Загальна вартість (UAH)</th>
                    <th>Статус Контракту</th>
                    <th>Дії</th>
                </tr>
            </thead>
            <tbody>
                {{#each contracts}}
                <tr>
```

```
                    <td>{{_id}}</td>
                    <td>{{postedDate}}</td>
                    <td>{{date}}</td>
                    <td>{{totalCost}}</td>
                    <td>
                                                <span class="new badge {{#if (eq statusCode "
SignedByCompany")}}red{{else}}blue{{/if}}"
                            data-badge-caption="{{statusText}}"></span>
                        {{#if isOverdue}}<span class="new badge red"
                            data-badge-caption="Протермінована доставка"></span>{{/if}}
                    </td>
                    <td>
                        {{#if (eq statusCode "SignedByCompany")}}
                           <a href="/provider-cabinet/contract/{{_id}}" class="btn-small red darken-
1">Підписати</a>
                        {{else}}
                        <a href="/provider-cabinet/contract/{{_id}}" class="btn-small grey">Деталі</a>
                        {{/if}}
                    </td>
                </tr>
                {{/each}}

                {{#unless contracts}}
                <tr>
                        <td colspan="6" class="center-align">Наразі немає активних контрактів, які
потребують Вашої уваги.
                    </td>
                </tr>
                {{/unless}}
            </tbody>
        </table>
    </div>
</div>

<style>
    .provider-cabinet-table,
    .provider-cabinet-table td,
    thead th {
        text-align: center;
    }
    .small-text {
        font-size: 0.8em;
```

```
        color: #9e9e9e;
            margin-top: 5px;
        }
        .card-content hr {
            border-color: rgba(255, 255, 255, 0.2);
        }
</style>
```

ДОДАТОК В

Демонстраційний матеріал

№ докумен-та	Позначення	Найменування	Додаткові відомості
		Текстові документи	
1	ГЮИК.506100.018 ПЗ	Пояснювальна записка	100 с., ф. А4
2		Апробація наукових	9 с., ф. А4
		результатів дослідження	
3		Лістинг програми	13 с., ф. А4
		Додаткові матеріали	
4		Демонстраційний матеріал	20 с., ф. А4

Змін.	Арк.	Номер докум.	Підпис	Дата	ГЮИК.506100.018 ВД		
Розроб.		Олінкевич Я.В.			Розроблення підсистеми автоматизації для взаємодії з постачальниками на промисловому підприємстві.	Літера	Аркуш	Аркушів
Перевір.		Хрустальова С.В.				Н	1	1
Н.контр.		Стародубцев М.Г.			Кафедра КІТАР ХНУРЕ			
Затв.		Невлюдов І.Ш.						