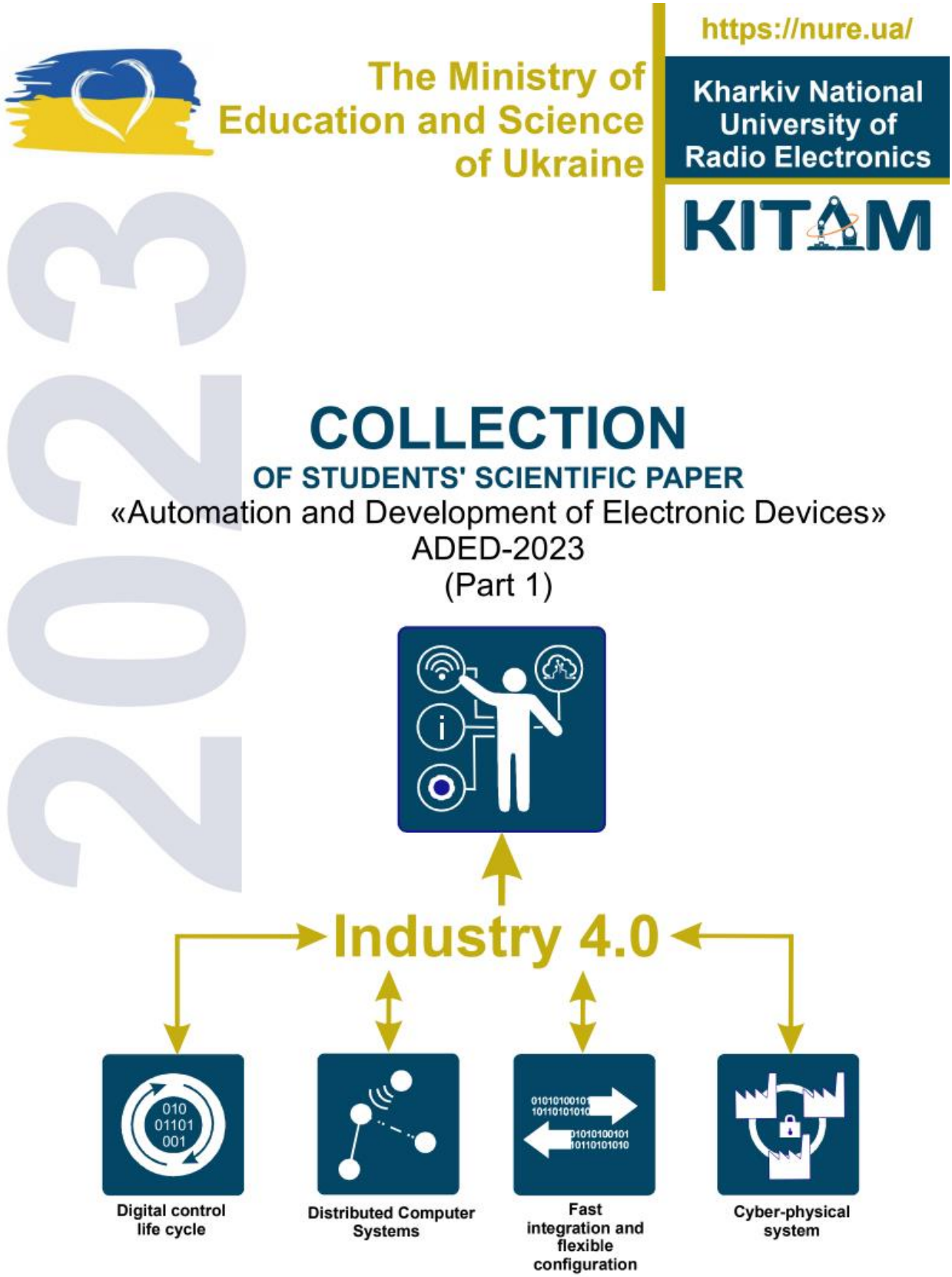
ДОДАТОК А

Апробація результатів наукових досліджень



https://nure.ua/

кафедра
Комп'ютерно-інтегрованих
технологій, автоматизації та мехатроніки

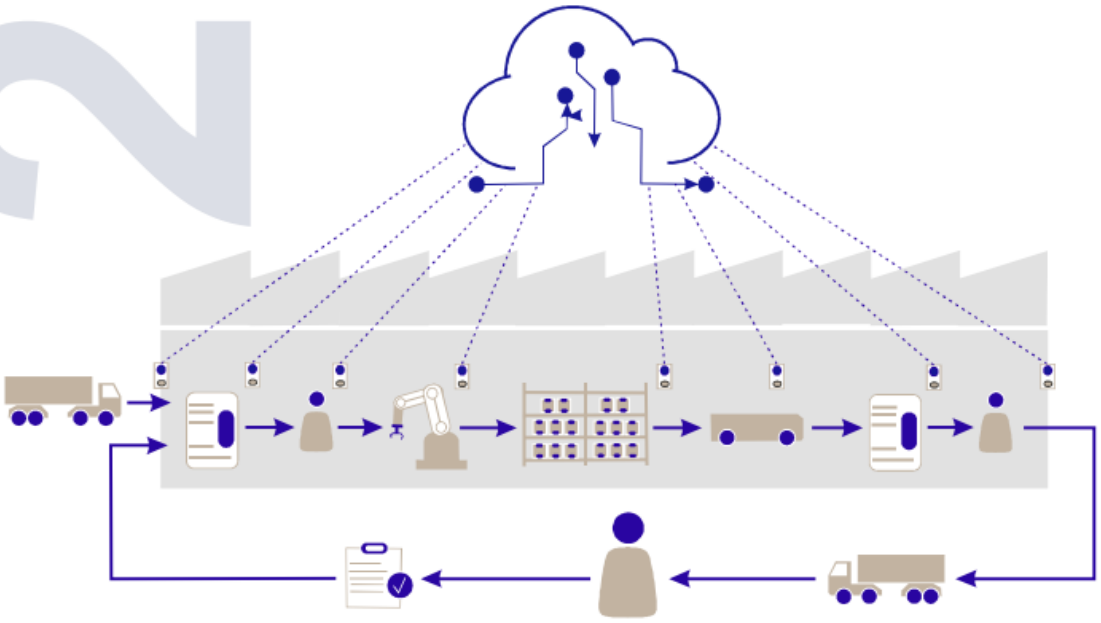
2023

ХНУРЕ

ЗБІРНИК

студентських наукових статей

«Автоматизація та приладобудування»
ADED-2023
(Випуск 1)
[електронне видання]



Industry 4.0

Автоматизація та Приладобудування («Automation and Development of Electronic Devices» ADED-2023) [Електронний ресурс] : збірник студентських наукових статей / Харківський національний університет радіоелектроніки ; [редкол.: І.Ш. Невлюдов та ін.]. – Харків : ХНУРЕ, 2023. – Вип. 1. – 336с.

Collection of Students' Scientific Paper «Automation and Development Of Electronic Devices» ADED-2023 Part 1 (Key infrastructure 2023) - Kharkiv/ The Editorial.: Nevlyudov I.Sh. (head), that all. Kharkiv: Kind of Kharkiv National Universitety of Radio Elektronics [electronic edition], 2023. – 336p 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 № 6 від 01.05.2023

Збірник містить наукові статті здобувачів першого (бакалаврського), другого (магістерського) рівнів вищої освіти кафедри комп'ютерно-інтегрованих технологій, автоматизації та мехатроніки (КІТАМ) 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, 172 Телекомунікації та радіотехніка, 171 Електроніка та 163 Біомедична інженерія. Статті надані в авторській редакції.

ЗМІСТ

УДК 658.51

АНАЛІЗ АВТОМАТИЗОВАНИХ СИСТЕМ КЕРУВАННЯ ТЕХНОЛОГІЧНИМИ ПРОЦЕСАМИ СУЧАСНОГО ПІДПРИЄМСТВА

С. В. Кравченко
Харківський національний університет радіоелектроніки
Україна, Харків, пр. Науки. 14
E-mail: serhii.kravchenko@nure.ua

Анотація: Дослідження присвячено аналізу автоматизованих систем сучасного підприємства з метою розробки таких системи та їх покращення. Розглядається Індустрія 5.0, її переваги та вимоги до її реалізації. Розкривається поняття кібер-фізичної системи. Кібер-фізичної система використовується в землеробстві, охороні здоров'я, транспортній сфері, моніторингу води, аерокосмічній промисловості та інших галузях. Вона складається з сенсорного модуля, модуля керування даними, модуля обробки та модуля приводів. Всі модулі з'єднанні інтернетом наступного покоління. Також визначаються основні завдання сучасних систем моніторингу технологічних процесів та принципи їх побудови.

Ключові слова: модуль; моніторинг; система; технологія.

ANALYSIS OF AUTOMATED SYSTEMS OF THE MODERN ENTERPRISE

S. Kravchenko
Kharkiv National University of Radio Electronics
Ukraine, 61166, Kharkiv, Nauky av.,14
E-mail: serhii.kravchenko@nure.ua

Abstract: The study is devoted to the analysis of automated systems of a modern enterprise with the aim of developing such systems and improving them. Industry 5.0, its advantages and requirements for its implementation are considered. The concept of a cyber-physical system is revealed. The cyber-physical system is used in agriculture, health care, transportation, water monitoring, the aerospace industry and other industries. It consists of a sensor module, a data management module, a processing module and an actuator module. All modules are connected to the Internet of the next generation. The main tasks of modern monitoring systems of technological processes and the principles of their construction are also determined.

Keywords: Module; Monitoring; System; Technology.

АКТУАЛЬНІСТЬ РОБОТИ. Автоматизовані системи управління технологічними процесами є актуальними напрямом розвитку сучасного підприємства, так як вони дозволяють значно покращити якість та ефективність виробництва. Такі системи знижують витрати за рахунок оптимізації процесів. Контроль над усіма етапами виробництва забезпечує зменшення помилок, що сприяє зменшенню відходів та обслуговування обладнання. Також автоматизовані системи підвищують безпеку на виробництві, що є важливою характеристикою. В даній статті розглядається розвиток таких систем та сучасні технології, які в них використовуються.

З першою промисловою революцією людина зрозуміла потенційне застосування технологій як засіб прогресу. Парові машини, складальні лінії та обчислювальні машини є одними з найважливіших досягнень, що виникли за останні декілька століть і всі вони направлені на створення технологій, які стають все краще, підвищуючи продуктивність та ефективність. Індустрія 5.0 приносить революцію, оскільки вона акцентує увагу на технології та передбачає, що справжній потенціал для прогресу відбувається внаслідок еволюції між людьми та машинами.

Індустрія 5.0 – це нова виробнича модель, в якій основна увага приділяється взаємодії людей і машин.

Етап, який передував Індустрії 5.0, ознаменувався появою цифрової індустрії. Такі досягнення, як Інтернет речей або поєднання штучного інтелекту та Big data, породили новий тип технології, яка може запропонувати компаніям знання, що базуються на даних.

Це, у свою чергу, трансформувалося в такі процеси, як операційна розвідка та бізнес-аналітика, які генерують моделі, що застосовують технології з метою прийняття все більш точних і менш невизначених рішень.

Однак на цьому етапі в Індустрії 4.0 мета полягала в тому, щоб мінімізувати участь людини і розставити пріоритети в автоматизації процесів. Певною мірою люди були поставлені в становище, коли вони конкурували з машинами. У випадку з Індустрією 5.0 ця тенденція зворотна: мета полягає в тому, щоб знайти баланс, при якому взаємодія машини та людини може запропонувати найбільші вигоди.

Зміни, розпочаті Індустрією 5.0, вже незворотні. Цей процес пропонує компаніям можливості все більш потужних машин у поєднанні з краще підготовленими експертами для сприяння ефективному, стійкому та безпечному виробництву.

Індустрія 5.0 – це новий спосіб розуміння виробництва, який має виробничі, економічні та комерційні наслідки. Тому компанії, які не адаптують своє виробництво до заводської моделі 5.0, скоро застаріють і не зможуть скористатися конкурентними перевагами, які вона може запропонувати.

Переваги Індустрії 5.0 [1]:

– Оптимізація витрат. Індустрія 5.0 бере на себе минулі поліпшення, які з часів Першої промислової революції породили ефективніші процеси. Пошук бізнес-моделей, які використовують найменші ресурси для отримання найвищого прибутку, знаходить у Індустрії 5.0 їх найвищий рівень досконалості на сьогоднішній день, оскільки людина та машина працюють разом, щоб приймати найкращі фінансові рішення для компанії.

– Більш екологічні рішення. Жодна з вищезгаданих промислових перетворень не була зосереджена на захисті навколишнього середовища як пріоритет. З Індустрією 5.0 нові корпоративні технології змінюють цю тенденцію. Це призвело до появи стійкої політики, в рамках якої, мінімізація відходів та управління ними стають найважливішими процесами, що робить організацію більш ефективною.

– Персоналізація та креативність. Технологічні інновації не дозволяють забезпечити такий ступінь персоналізації, що відповідає потребам клієнтів. Персонал, який є частиною Індустрії 5.0, використовуватиме потенціал технологій, але також знайде можливість додати свої власні ідеї, які призведуть до продукту, розробленого з урахуванням персоналізації. Крім того, автоматизація, досягнута під час Індустрії 4.0, дозволяє працівникам звільнитися від певних завдань, що повторюються, зосередившись на розробці більш потужних стратегій або застосуванні своєї творчості.

Вимоги Індустрії 5.0:

– Навчений персонал. Індустрія 5.0 принесла роль директора з робототехніки. Ця людина спеціалізується на взаємодії між машинами та операторами, а також володіє знаннями в таких галузях, як робототехніка та штучний інтелект. Його роль компанії передбачає прийняття рішень навколо цих чинників. Навчання співробітників також зробить стрибок уперед з поширенням віртуальної освіти. Це дозволяє знизити витрати для компаній, оскільки не вимагає зупинення виробництва для навчання своїх працівників. Крім того, це також призводить до безпечнішого навчання, яке запобігає схильності працівників до непотрібних ризиків під час навчання. Комунікація та мотивація співробітників також підвищуються завдяки інтерактивним навчальним середовищам. Також очікується генерація безлічі робочих місць, пов'язаних із взаємодією з роботизованими системами та штучним інтелектом, серед інших технологій.

– Правильна технологія. Термін "коботи" був придуманий стосовно Індустрії 5.0: колаборативні роботи, призначені для простої та інтуїтивно зрозумілої взаємодії з людьми. Певним чином вони виступають у ролі учнів, здатних спостерігати за діями людини та

відтворювати їх, допомагаючи операторам. Розширення Digital Twins також стане ще однією необхідною технологією на виробництві Індустрії 5.0. Це візуальні моделі продукту або процесу, і їх генерація дозволяє краще зрозуміти та протестувати їх. Крім того, поява все більш складних процесів вимагатиме відповідного програмного забезпечення, здатного керувати цим величезним обсягом даних і надавати операторам-людям простір, який вони можуть використовувати для взаємодії з машинами. Платформа Nexus Integra – це програмне забезпечення, яке призведе до трансформації промислової компанії в Індустрію 5.0. Це інтегрована система для масштабного управління промисловими активами, що дозволяє компаніям зробити стрибок до цифрової трансформації.

Індустрія 4.0 принесла сучасним підприємствам кібер-фізичних системи. Кібер-фізичні системи (CPS) є інтеграцією обчислень, мереж і фізичних процесів [2]. Вбудовані комп'ютери та мережі контролюють фізичні процеси за допомогою контурів зворотного зв'язку де фізичні процеси впливають на обчислення та навпаки. Економічний і соціальний потенціал таких систем значно вищий за те, що було реалізовано, тому великі інвестиції робляться по всьому світу для розвитку цих систем. CPS інтегрує динаміку фізичних процесів з динамікою програмного забезпечення та мережі, надаючи абстракції та методи моделювання, проектування та аналізу.

CPS можна використовувати в широкому діапазоні сфер застосування, включаючи інтелектуальний транспорт, землеробство, охорону здоров'я, моніторинг води, аерокосмічну промисловість тощо. Розглянемо деякі з них більш детально.

Транспортна кібер-фізична система (VCPS) – це ефективна та точна інтегрована система управління транспортом, яка має працювати в режимі реального часу. На основі сучасних технологій, таких як електроніка, комп'ютери, датчики та мережі, традиційні види транспорту стають розумнішими. Зі збільшенням кількості особистих автомобілів багатьом проблемам, такі як затори, забруднення повітря та проблеми безпеки, приділяють більше уваги. Просунуті обчислювальні та сенсорні можливості, такі як повітряний, залізничний та автомобільний контроль з метою підвищення безпеки і пропускної здатності, будуть широко використовуватися в транспортній системи наступного покоління. Прикладом таких систем є самостійна швидкісна система «AHS» (Automated Highway system). Її розвиток спрямований на досягнення більш безпечного та інтелектуального трафіку. Для таких систем необхідні висока обчислювальна потужність для комплексного управління трафіком, алгоритми, які обчислюють, наприклад, найкращий маршрут відповідно до дорожньої ситуації.

Кібер-фізична система для землеробства реалізує повний комплекс сучасних систем, стратегії та технології ведення сільського господарства. Проектування землеробства включає управління даними, виробничі досліди, фундаментальну географічну інформацію сільськогосподарських угідь, інформацію про мікроклімат та інші дані. Прикладом даної системи є проект «Underground wireless sensor network». В даній мережі сенсорні вузли системи обмінюються інформацією бездротово через ґрунт.

Кібер-фізичні системи, які використовуються в охороні здоров'я (HCPS) замінить традиційні пристрої в цій сфері. З датчиків та мереж, різні пристрої, працюючи разом, будуть виявляти фізичний стан пацієнтів в режимі реального часу. Особливо це важливо для критичних пацієнтів зі захворюваннями серця. Портативні термінальні пристрої, які носять пацієнти, можуть виявити стан пацієнта в будь-який час і надсилати своєчасне попередження або прогноз наперед. Крім того, співпраця між цими системами та доставка даних у режимі реального часу було б багато зручніше для пацієнтів. CPS для охорони здоров'я та медичного обладнання вимагають нове покоління аналізу, синтезу та інтеграційних технологій.

Архітектура кібрс-фізичних систем складається з 5 основних модулів, що поєднуються Інтернетом наступного покоління [3]. Схематично вона представлена на рисунку 1.

Сенсорний модуль необхідний для збору даних із фізичного світу за допомогою датчиків. Основна функція роботи цього модуля це обізнаність навколишнього середовища, яка

досягається шляхом попередньої обробки даних. Вихідні дані надаються модулю управління даними.

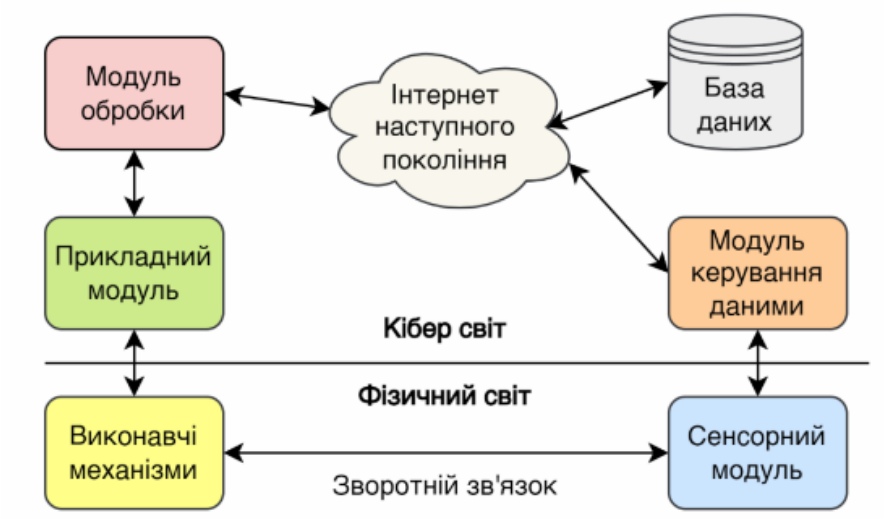


Рисунок 1 – Архітектура кібер-фізичної системи

Модуль керування даними складається з обчислювальних пристроїв і носіїв інформації. Це забезпечує неоднорідну обробку даних, таку як нормалізація, зменшення шуму, зберігання даних та інші подібні функції. Модуль розглядається як міст між динамічним середовищем і сервісами, оскільки він збирає дані від датчиків і пересилає дані модулям, які відповідають за обслуговування, використовуючи Інтернет наступного покоління.

Загальною рисою Інтернету наступного покоління є можливість для програм вибирати шлях або шляхи, якими їхні пакети проходять між джерелом і одержувачем. Цей динамічний характер інтернет-сервісу необхідний для розробки кібер-фізичної системи. На відміну від поточної архітектури Інтернету, де протоколи маршрутизації знаходять єдиний (найкращий) шлях між джерелом і одержувачем, майбутні протоколи маршрутизації Інтернету повинні будуть надавати програмам можливість вибору шляхів.

Модуль обробки забезпечує типові функції всієї системи, включаючи прийняття рішень, аналіз завдань, розклад завдань тощо. Після отримання даних цей модуль розпізнає та надсилає дані до доступних сервісів.

Прикладний модуль надає певну функціональність для користувача або іншого модуля. Він може використовувати дані з сенсорів та приводів для контролю фізичного світу або надання інформації про нього. Також модуль призначений для виконання координації дій або обміну даними.

Виконавчі механізми та датчики є двома різними електронними пристроями, які взаємодіють із фізичним середовищем. Виконавчий механізм отримує команди від прикладного модуля та виконує їх.

Зворотній зв’язок служить для мінімізації обробки даних шляхом зв’язку між датчиком і приводом для безпосереднього виконання необхідних дій.

Частина забезпечення безпеки за своєю суттю є важливою для всієї системи. Безпеку CPS можна розділити на наступні три етапи:

– безпека інформованості, яка полягає в забезпеченні безпеки та точності інформації, зібраної з фізичного середовища;

– транспортна безпека, яка полягає у запобіганні знищенню даних під час процесів передачі;

– фізична безпека, що представляє з себе процедури безпеки на серверах або робочих станціях.

Важливою частиною в сучасних системах автоматизації є моніторинг за технологічними процесами. Він повинен розв'язувати такі завдання [4]:

– збір технологічних даних у реальному часі;

– збереження та попереднє опрацювання даних з використанням хмарних технологій;

– аналітичне та інтелектуальне опрацювання даних;

– візуалізація накопичених даних про технологічні процеси і подання результатів опрацювання даних у вигляді графіків і діаграм;

– формування сигналів управління для виконавчих механізмів;

– прийняття управлінських рішень для управління технологічними процесами;

– формування звітів про стан технологічного процесу.

Моніторинг має дві складові: апаратну та програмну.[5-7]

В апаратну частину входять безпосередньо засоби для накопичення інформації про технологічний процес – давачі, а також пристрої, які будуть отримувати зібрану інформацію, обробляти її та передавати на інші рівні управління. Використовуються як дротові, так і бездротові технології для взаємодії між цими компонентами. Апаратні компоненти, які використовують для збору та попередньої обробки даних, повинні забезпечувати розв'язання задач у реальному часі. Крім цього, є ще вимоги, які необхідно врахувати під час вибору апаратних компонентів: масогабаритні характеристики, оскільки комп'ютерні компоненти можуть бути розміщені безпосередньо біля давачів і виконавчих механізмів, а також високі вимоги до споживаної потужності, живучості та надійності.

Програмна складова рішення повинна стабільно підтримувати високонавантажені обчислення, працювати за принципом асинхронності та бути побудованою з використанням відкритого програмного забезпечення.[8-10]

Розробляючи рішення моніторингу, потрібно звернути увагу на вже готові апаратні та програмні засоби, оскільки розроблення та виготовлення нових потребує значних коштів і часу. Під час вибору компонентів необхідно враховувати інформацію про готові апаратно-програмні компоненти, їх технічні характеристики, відповідність інтерфейсів стандартам, можливості їх покупки тощо.

Розроблення засобів моніторингу повинно ґрунтуватися на таких принципах:

– інтеграції комп'ютерних, комунікаційних і програмних компонентів;

– модульності, який передбачає використання функціонально завершених компонентів з виходом на стандартний інтерфейс;

– відкритості, за яким запропоноване рішення передбачає можливості нарощування та оновлення функцій;

– сумісності, яка передбачає використання стандартних провідних і безпровідних інтерфейсів для зв'язку між компонентами.

Використання автоматизованих систем управління технологічними процесами є актуальним в сучасних підприємствах, так як вони дозволяють покращити ефективність та якість виробництва, знизити витрати та забезпечити безпеку. Вже зараз в багатьох сферах використовують кібер-фізичні системи, що можуть аналізувати та контролювати всі технологічні процеси. Наступним етапом розвитку виробництва є Індустрія 5.0 і цей процес вже не зворотній. Цей етап розвитку пропонує створити симбіоз роботи автоматизованих систем та людини для отримання найкращих результатів. Такий підхід вирішить проблему конкуренції людей та машин в виробництві, яка є зараз. Аналізуючи це, можна сказати, що виробничу галузь очікує цікаве майбутнє, яке настане досить скоро, взявши до уваги сучасні технології та швидкість їх розвитку.[11-13]

ЛІТЕРАТУРА

1. Industry 5.0: the new revolution. NexusIntegra [Електронний ресурс]. – режим доступу: https://nexusintegra.io/industry-5-0-the-new-revolution

2. Cyber-Physical Systems [Електронний ресурс]. – режим доступу: https://ptolemy.berkeley.edu/projects/cps

3. Syed Hassan Ahmed, Gwanghyeon Kim, Dongkyun Kim Cyber Physical System: Architecture, applications and research challenges. School of Computer Science & Engineering, Kyungpook National University, Daegu, Korea / – 2013, – С. 3-4. [Електронний ресурс]. – режим доступу: https://www.researchgate.net/publication/261279251_Cyber_Physical_System_Architecture_applications_and_research_challenges

4. І. Г. Цмоць, А. Є. Батюк, А. В. Яворський, Т. В. Теслюк Система моніторингу технологічних процесів «Розумного підприємства». Національний університет «Львівська політехніка», кафедра автоматизованих систем управління / – 2018, – С. 2-3. [Електронний ресурс]. – режим доступу: https://science.lpnu.ua/sites/default/files/journal-paper/2019/jan/15441/10-17.pdf

5. Vladyslav, Y., & Bronnikov, A. (2020, October). ANALYSIS OF THE CMMI MODEL APPLICATION FOR SOLVING THE TASKS OF CPPS CONTROL PROCESSES AUTOMATION DEVELOPMENT. In The 4 th International scientific and practical conference "Actual trends of modern scientific research"(October 11-13, 2020) MDPC Publishing, Munich, Germany. 2020. 386 p. (p. 128).

6. Yevsieiev, V. V., & Bronnikov, A. I. (2020). Development of databases interconnection "essences" information model for cyber-physical production systems additive cyber design creation automation. Збірник Наукових Праць НУК, №3. С.56-62. DOI https://doi.org/10.15589/znp2020.3(481).7

7. Nevliudov I., Omarov M., Yevsieiev V., Bronnikov A., Lyashenko V. Method of Algorithms for Cyber-Physical Production Systems Functioning Synthesis // International Journal of Emerging Trends in Engineering Research. – 2020. – Vol. 8(10). – P. 7465-7473.

8. Yevsieiev V., Bronnikov A. Information systems development methodologies application analysis for cyber-physical production systems development. III International scientific-practical conference "Theory, science and practice" (Japan, Tokyo, 5–8 October 2020). P. 398–401. DOI: 10.46299/ISG.2020.II.III.

9. Yevsieiev V., Bronnikov A. Analysis of the cyber-physical production systems implementation impact to achieve the goals of lean production. The IIth International scientific and practical conference «Development of scientific and practical approaches in the era of globalization» (USA, Boston, 28–30 September. 2020). P.221–226. DOI:10.46299/ISG.2020.II.II.

10. Невлюдов І.Ш. Автоматизована система керування технологічними процесами в SCADA системі TRACE MODE 6: Навчальний посібник / І.Ш. Невлюдов, А.О. Андрусевич, В.В. Євсєєв, С.С. Максимова, М.Г. Стародубцев, В.В.Невлюдова. Кривий Ріг: Криворізький коледж НАУ, 2018. 320 с.

11. Khalid, M. S., Yevsieiev, V., Nevliudov, I. S., Lyashenko, V., & Wahid, R. (2022). HMI Development Automation with GUI Elements for Object-Oriented Programming Languages Implementation. International Journal of Engineering Trends and Technology, 70.1, 139-145.

12. Nevliudov, I., & et al.. (2021). GUI Elements and Windows Form Formalization Parameters and Events Method to Automate the Process of Additive CyberDesign CPPS Development. Advances in Dynamical Systems and Applications, 16(2), 441-455.

13. Моделі та методи кіберфізичних виробничих систем в концепції Industry 4.0 : монографія / І. Ш. Невлюдов, В. В. Євсєєв, А. О. Андрусевич, С. С. Максимова ; – Oktan Print – Prague. 2023. – 321 с.

Науковий керівник: Бронніков Артем Ігорович, доцент кафедри комп'ютерно-інтегрованих технологій, автоматизації та мехатроніки, Харківський національний університет радіоелектроніки

ДОДАТОК Б

Текст програми

AuthService.csproj

```
<Project Sdk="Microsoft.NET.Sdk.Web">

  <PropertyGroup>
    <TargetFramework>net7.0</TargetFramework>
    <Nullable>enable</Nullable>
    <ImplicitUsings>enable</ImplicitUsings>
    <RootNamespace>MrMonitor.$(MSBuildProjectName.Replace(" ",
"_"))</RootNamespace>
    <PackageId>MrMonitor.$(AssemblyName)</PackageId>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="Microsoft.AspNetCore.Authentication.JwtBearer"
Version="7.0.11" />
    <PackageReference Include="MySql.EntityFrameworkCore" Version="7.0.5" />
  </ItemGroup>

  <ItemGroup>
    <ProjectReference Include="..\Common\Common.csproj" />
  </ItemGroup>

</Project>
```

User.cs

```
namespace MrMonitor.AuthService.Models.Db;

public class User
{
    public long Id { get; set; }

    public string Firstname { get; set; }

    public string? Lastname { get; set; }

    public string Username { get; set; }

    public string Password { get; set; }
}
```

AuthDbContext.cs

```
using Microsoft.EntityFrameworkCore;
using MrMonitor.AuthService.Models.Db;

namespace MrMonitor.AuthService;

public class AuthDbContext : DbContext
{
    public AuthDbContext(DbContextOptions<AuthDbContext> options) :
base(options)
    {
        Database.EnsureCreated();
    }

    public DbSet<User> Users { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<User>().HasKey(u => u.Id);

        modelBuilder.Entity<User>().Property(u => u.Username).IsRequired();
        modelBuilder.Entity<User>().HasIndex(u => u.Username).IsUnique();

        modelBuilder.Entity<User>().Property(u => u.Firstname).IsRequired();
        modelBuilder.Entity<User>().Property(u =>
u.Lastname).HasDefaultValue(null);

        modelBuilder.Entity<User>().Property(u => u.Password).IsRequired();
    }
}
```

UserStore.cs

```
using Common.Models;
using MrMonitor.AuthService.Models.Db;
using Microsoft.EntityFrameworkCore;
using System.Linq.Expressions;
using MrMonitor.AuthService.Enums;

namespace MrMonitor.AuthService.Stores;

public class UserStore : IUserStore
{
    private const string UserAlreadyExistsMessagePattern = "User with username
'{0}' already exists.";
    private const string UserNotFoundMessage = "User not found.";

    private readonly AuthDbContext _dbContext;

    public UserStore(AuthDbContext dbContext)
    {
        _dbContext = dbContext ?? throw new
ArgumentNullException(nameof(dbContext));
    }

    public async Task<Result> AddUserAsync(User user)
    {
        ArgumentNullException.ThrowIfNull(user);

        try
        {
            var findUserResult = await FindUserByUsernameAsync(user.Username);

            if (findUserResult.IsSuccess)
            {
                return Result.FromFail(
                    (int)ResultStatus.UserAlreadyExists,
                    string.Format(UserAlreadyExistsMessagePattern,
user.Username));
            }

            await _dbContext.Users.AddAsync(user);
            await _dbContext.SaveChangesAsync();

            return Result.FromSuccess();
        }
        catch (Exception e)
        {
            return Result.FromFail((int)ResultStatus.ProcessingError,
e.Message);
```

```
        }
    }

    public async Task<Result> UpdateUserAsync(User updatedUser)
    {
        ArgumentNullException.ThrowIfNull(updatedUser);

        try
        {
            var findUserResult = await FindUserByIdAsync(updatedUser.Id);

            if (!findUserResult.IsSuccess)
            {
                return Result.FromFail(findUserResult.Status.Value,
findUserResult.Message);
            }

            var user = findUserResult.Entity;

            findUserResult = await
FindUserByUsernameAsync(updatedUser.Username);

            if (!findUserResult.IsSuccess && findUserResult.Status !=
(int)ResultStatus.UserNotFound)
            {
                return Result.FromFail(findUserResult.Status.Value,
findUserResult.Message);
            }

            if (findUserResult.IsSuccess && user.Id !=
findUserResult.Entity.Id)
            {
                return Result.FromFail(
                    (int)ResultStatus.UserAlreadyExists,
                    string.Format(UserAlreadyExistsMessagePattern,
user.Username));
            }

            _dbContext.Entry(user).CurrentValues.SetValues(updatedUser);
            _dbContext.Entry(user).Property(x => x.Password).IsModified =
false;

            await _dbContext.SaveChangesAsync();

            return Result.FromSuccess();
        }
        catch (Exception e)
        {
            return Result.FromFail((int)ResultStatus.ProcessingError,
e.Message);
        }
```

```
    }

    public async Task<Result> ChangeUserPasswordAsync(long? userId, string
newPassword)
    {
        ArgumentNullException.ThrowIfNull(userId);
        ArgumentException.ThrowIfNullOrEmpty(newPassword);

        try
        {
            var findUserResult = await FindUserByIdAsync(userId);

            if (!findUserResult.IsSuccess)
            {
                return Result.FromFail(findUserResult.Status.Value,
findUserResult.Message);
            }

            findUserResult.Entity.Password = newPassword;

            await _dbContext.SaveChangesAsync();

            return Result.FromSuccess();
        }
        catch (Exception e)
        {
            return Result.FromFail((int)ResultStatus.ProcessingError,
e.Message);
        }
    }

    public async Task<Result<User>> FindUserByUsernameAsync(string username)
    {
        ArgumentException.ThrowIfNullOrEmpty(username);

        return await FindUserAsync(u => u.Username == username);
    }

    public async Task<Result<User>> FindUserByIdAsync(long? userId)
    {
        ArgumentNullException.ThrowIfNull(userId);

        return await FindUserAsync(u => u.Id == userId);
    }

    private async Task<Result<User>> FindUserAsync(Expression<Func<User, bool>>
predicate)
    {
        try
        {
            var user = await _dbContext.Users.FirstOrDefaultAsync(predicate);
```

```
            if (user == null)
            {
                return Result<User>.FromFail((int)ResultStatus.UserNotFound,
UserNotFoundMessage);
            }

            return Result<User>.FromSuccess(user);
        }
        catch (Exception e)
        {
            return Result<User>.FromFail((int)ResultStatus.ProcessingError,
e.Message);
        }
    }
}
```

AuthController.cs

```
using System.Text;
using System.Security.Claims;
using System.IdentityModel.Tokens.Jwt;
using Microsoft.AspNetCore.Mvc;
using Microsoft.IdentityModel.Tokens;
using MrMonitor.AuthService.Models.Db;
using MrMonitor.AuthService.Stores;
using MrMonitor.Common.Options;
using Microsoft.AspNetCore.Authorization;
using MrMonitor.AuthService.Models.Requests;
using MrMonitor.AuthService.Models.Responses;
using System.ComponentModel.DataAnnotations;
using MrMonitor.AuthService.Enums;

namespace MrMonitor.AuthService.Controllers;

[ApiController]
[Route("[controller]")]
public class AuthController : Controller
{
    private const string InvalidUserCredentialsMessage = "Invalid username or
password.";
    private const string UnauthorizedMessage = "Unauthorized.";
    private const string BadJwtTokenErrorMessage = "Bad JWT token. Please
relogin.";
    private const string ServerErrorMessage = "Server error.";

    private readonly IUserStore _userStore;
    private readonly AuthOptions _authOptions;

    public AuthController(IUserStore userStore, AuthOptions
jwtAuthConfiguration)
    {
        _userStore = userStore;
        _authOptions = jwtAuthConfiguration;
    }

    [HttpPost]
    [Route("login")]
    public async Task<IActionResult> LoginAsync(
        [FromBody, Required] LoginRequest request)
    {
        var findUserResult = await
_userStore.FindUserByUsernameAsync(request.Username);

        if (!findUserResult.IsSuccess)
        {
```

```
            return findUserResult.Status == (int)ResultStatus.UserNotFound
                ? Unauthorized(InvalidUserCredentialsMessage)
                : StatusCode(StatusCodes.Status500InternalServerError,
ServerErrorMessage);
        }

        return findUserResult.Entity.Password != request.Password
            ? Unauthorized(InvalidUserCredentialsMessage)
            : Ok(GetJwtToken(findUserResult.Entity));
    }

    [HttpPut]
    [Route("createAccount")]
    public async Task<IActionResult> CreateAccountAsync(
        [FromBody, Required] CreateAccountRequest request)
    {
        var newUser = new User
        {
            Username = request.Username,
            Firstname = request.Firstname,
            Lastname = request.Lastname,
            Password = request.Password
        };

        var addUserResult = await _userStore.AddUserAsync(newUser);

        if (addUserResult.IsSuccess)
        {
            return Ok(GetJwtToken(newUser));
        }

        return addUserResult.Status == (int)ResultStatus.UserAlreadyExists
            ? BadRequest(addUserResult.Message)
            : StatusCode(StatusCodes.Status500InternalServerError,
ServerErrorMessage);
    }

    [Authorize]
    [HttpPut]
    [Route("updateUserInfo")]
    public async Task<IActionResult> UpdateUserInfoAsync(
        [FromBody, Required] UpdateUserInfoRequest request)
    {
        var userId = GetUserIdFromClaims();

        if (!userId.HasValue)
        {
            return BadRequest(BadJwtTokenErrorMessage);
        }

        var updatedUser = new User
```

```
        {
            Id = userId.Value,
            Username = request.Username,
            Firstname = request.Firstname,
            Lastname = request.Lastname
        };

        var updateUserResult = await _userStore.UpdateUserAsync(updatedUser);

        if (updateUserResult.IsSuccess)
        {
            return Ok();
        }

        return updateUserResult.Status switch
        {
            (int)ResultStatus.UserAlreadyExists =>
BadRequest(updateUserResult.Message),

            (int)ResultStatus.UserNotFound =>
NotFound(updateUserResult.Message),

            _ => StatusCode(StatusCodes.Status500InternalServerError,
ServerErrorMessage)
        };
    }

    [Authorize]
    [HttpPut]
    [Route("changePassword")]
    public async Task<IActionResult> ChangeUserPasswordAsync(
        [FromBody, Required] ChangeUserPasswordRequest request)
    {
        var userId = GetUserIdFromClaims();

        if (!userId.HasValue)
        {
            return BadRequest(BadJwtTokenErrorMessage);
        }

        var findUserResult = await _userStore.FindUserByIdAsync(userId);

        if (!findUserResult.IsSuccess)
        {
            return findUserResult.Status == (int)ResultStatus.UserNotFound
                ? NotFound(findUserResult.Message)
                : StatusCode(StatusCodes.Status500InternalServerError,
ServerErrorMessage);
        }

        if (findUserResult.Entity.Password != request.Password)
```

```
        {
            return Unauthorized(UnauthorizedMessage);
        }

        var changeUserPasswordResult = await _userStore
            .ChangeUserPasswordAsync(userId.Value, request.NewPassword);

        if (changeUserPasswordResult.IsSuccess)
        {
            return Ok();
        }

        return StatusCode(StatusCodes.Status500InternalServerError,
ServerErrorMessage);
    }

    [Authorize]
    [HttpGet]
    [Route("getMe")]
    public async Task<IActionResult> GetMeAsync()
    {
        var userId = GetUserIdFromClaims();

        if (!userId.HasValue)
        {
            return BadRequest(BadJwtTokenErrorMessage);
        }

        var findUserResult = await _userStore.FindUserByIdAsync(userId);

        if (findUserResult.IsSuccess)
        {
            var user = findUserResult.Entity;

            return Ok(new GetMeResponse()
            {
                UserId = user.Id,
                Username = user.Username,
                Firstname = user.Firstname,
                Lastname = user.Lastname
            });
        }

        return findUserResult.Status == (int)ResultStatus.UserNotFound
            ? NotFound(findUserResult.Message)
            : StatusCode(StatusCodes.Status500InternalServerError,
ServerErrorMessage);
    }

    private string GetJwtToken(User user)
    {
```

```
        var tokenHandler = new JwtSecurityTokenHandler();

        var token = tokenHandler.CreateToken(new SecurityTokenDescriptor
        {
            Subject = new ClaimsIdentity(new[]
            {
                new Claim("userId", user.Id.ToString())
            }),
            Expires = DateTime.UtcNow.AddDays(14),
            Issuer = _authOptions.Issuer,
            Audience = _authOptions.Audience,
            SigningCredentials = new SigningCredentials(
                new
SymmetricSecurityKey(Encoding.UTF8.GetBytes(_authOptions.SecretKey)),
                SecurityAlgorithms.HmacSha512Signature)
        });

        return tokenHandler.WriteToken(token);
    }

    private long? GetUserIdFromClaims()
    {
        var userIdClaim = User.Claims.FirstOrDefault(x => x.Type == "userId");
        long userId;

        if (userIdClaim == null || !long.TryParse(userIdClaim.Value, out
userId))
        {
            return null;
        }

        return long.Parse(userIdClaim.Value);
    }
}
```

ServiceCollectionExtension.cs

```
using Microsoft.EntityFrameworkCore;
using MrMonitor.Common.Options;
using MrMonitor.AuthService.Stores;

namespace MrMonitor.AuthService.Extensions;

public static class ServiceCollectionExtension
{
    public static IServiceCollection AddAuthDbContext(this IServiceCollection
services, IConfiguration configuration)
    {
        ArgumentNullException.ThrowIfNull(services);
        ArgumentNullException.ThrowIfNull(configuration);

        services.AddDbContext<AuthDbContext>(options =>
        {
            options.UseMySQL(configuration.Get<DatabaseOptions>().ConnectionStr
ing);
        });

        return services;
    }

    public static IServiceCollection AddAuthStores(this IServiceCollection
services)
    {
        ArgumentNullException.ThrowIfNull(services);

        services.AddScoped<IUserStore, UserStore>();

        return services;
    }

    public static IServiceCollection AddAuthOptions(this IServiceCollection
services, IConfiguration configuration)
    {
        ArgumentNullException.ThrowIfNull(services);
        ArgumentNullException.ThrowIfNull(configuration);

        var authOptions = configuration.Get<AuthOptions>();

        services.AddSingleton(authOptions);

        return services;
    }
}
```

WorkerService.csproj

```
<Project Sdk="Microsoft.NET.Sdk.Web">

  <PropertyGroup>
    <TargetFramework>net7.0</TargetFramework>
    <Nullable>enable</Nullable>
    <ImplicitUsings>enable</ImplicitUsings>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="Microsoft.EntityFrameworkCore" Version="7.0.12"
/>
    <PackageReference Include="MySql.EntityFrameworkCore" Version="7.0.5" />
    <PackageReference Include="Redis.OM" Version="0.6.1" />
  </ItemGroup>

  <ItemGroup>
    <ProjectReference Include="..\Common\Common.csproj" />
  </ItemGroup>

</Project>
```

WorkNode.cs

```
namespace MrMonitor.WorkerService.Models.Db;

public class WorkNode
{
    public long Id { get; set; }

    public string Name { get; set; }

    public string Token { get; set; }

    public ICollection<UserWorkNode> UserWorkNodes { get; set; }

    public ICollection<WorkNodeStateData> WorkNodeStateData { get; set; }
}
```

UserWorkNode.cs

```
using MrMonitor.WorkerService.Enums;

namespace MrMonitor.WorkerService.Models.Db;

public class UserWorkNode
{
    public long Id { get; set; }

    public long WorkNodeId { get; set; }

    public long UserId { get; set; }

    public WorkNodePermission Permission { get; set; }

    public WorkNode WorkNode { get; set; }

    public ICollection<Message> Messages { get; set; }
}
```

WorkNodeStateData.cs

```
namespace MrMonitor.WorkerService.Models.Db;

public class WorkNodeStateData
{
    public long Id { get; set; }

    public long WorkNodeId { get; set; }

    public string Key { get; set; }

    public int EntriesCount { get; set; }

    public string Data { get; set; }

    public DateTime CreatedDateTime { get; set; }

    public WorkNode WorkNode { get; set; }
}
```

Message.cs

```
using MrMonitor.WorkerService.Enums;

namespace MrMonitor.WorkerService.Models.Db;

public class Message
{
    public long Id { get; set; }

    public long UserWorkNodeId { get; set; }

    public SenderType SenderType { get; set; }

    public DateTime DateTime { get; set; }

    public UserWorkNode UserWorkNode { get; set; }

    public ICollection<MessageContent> Contents { get; set; }
}
```

MessageContent.cs

```
using MrMonitor.WorkerService.Enums;

namespace MrMonitor.WorkerService.Models.Db;

public class MessageContent
{
    public long Id { get; set; }

    public long MessageId { get; set; }

    public MessageContentType Type { get; set; }

    public string Data { get; set; }

    public Message Message { get; set; }
}
```

WorkNodeState.cs

```
using Redis.OM.Modeling;

namespace MrMonitor.WorkerService.Models.Rdb
{
    [Document(StorageType = StorageType.Json, Prefixes = new[] {
"WorkNodeState" })]
    public class WorkNodeState
    {
        [RedisIdField]
        [Indexed]
        public string? Id { get; set; }

        [Indexed]
        public long WorkNodeId { get; set; }

        [Indexed]
        public string? Key { get; set; }

        [Indexed]
        public float Value { get; set; }

        [Indexed(Sortable = true)]
        public DateTime DateTime { get; set; }
    }
}
```

WorkerDbContext.cs

```
using Microsoft.EntityFrameworkCore;
using MrMonitor.WorkerService.Models.Db;

namespace MrMonitor.WorkerService;

public class WorkerDbContext : DbContext
{
    public WorkerDbContext(DbContextOptions<WorkerDbContext> options) :
base(options)
    {
        try
        {
            Database.EnsureCreated();
        }
        catch { }
    }

    public DbSet<WorkNode> WorkNodes { get; set; }

    public DbSet<WorkNodeStateData> WorkNodeStateData { get; set; }

    public DbSet<UserWorkNode> UserWorkNodes { get; set; }

    public DbSet<Message> Messages { get; set; }

    public DbSet<MessageContent> MessageContents { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<WorkNode>().HasKey(e => e.Id);
        modelBuilder.Entity<WorkNode>().HasIndex(e => e.Token).IsUnique();
        modelBuilder.Entity<WorkNode>().HasMany(e => e.UserWorkNodes).WithOne(e
=> e.WorkNode);
        modelBuilder.Entity<WorkNode>().HasMany(e =>
e.WorkNodeStateData).WithOne(e => e.WorkNode);

        modelBuilder.Entity<UserWorkNode>().HasKey(e => e.Id);
        modelBuilder.Entity<UserWorkNode>().HasIndex(e => new { e.WorkNodeId,
e.UserId }).IsUnique();
        modelBuilder.Entity<UserWorkNode>().HasMany(e => e.Messages).WithOne(e
=> e.UserWorkNode);
        modelBuilder.Entity<UserWorkNode>().Property(e =>
e.Permission).HasConversion<int>();

        modelBuilder.Entity<WorkNodeStateData>().HasKey(e => e.Id);

        modelBuilder.Entity<Message>().HasKey(e => e.Id);
```

```
        modelBuilder.Entity<Message>().HasMany(e => e.Contents).WithOne(e =>
e.Message);
        modelBuilder.Entity<Message>().Property(e =>
e.SenderType).HasConversion<int>();

        modelBuilder.Entity<MessageContent>().HasKey(e => e.Id);
        modelBuilder.Entity<MessageContent>().Property(e =>
e.Type).HasConversion<int>();
    }
}
```

ChatStore.cs

```
using Common.Models;
using Microsoft.EntityFrameworkCore;
using MrMonitor.WorkerService.Enums;
using MrMonitor.WorkerService.Models.Db;
using Redis.OM;

namespace MrMonitor.WorkerService.Stores;

public class ChatStore : IChatStore
{
    private readonly WorkerDbContext _dbContext;

    public ChatStore(WorkerDbContext dbContext)
    {
        _dbContext = dbContext ?? throw new
ArgumentNullException(nameof(dbContext));
    }

    public async Task<Result> AddMessageAsync(Message message)
    {
        ArgumentNullException.ThrowIfNull(message);

        try
        {
            await _dbContext.Messages.AddAsync(message);
            await _dbContext.SaveChangesAsync();

            return Result.FromSuccess();
        }
        catch (Exception e)
        {
            return Result.FromFail((int)ResultStatus.ProcessingError,
e.Message);
        }
    }

    public async Task<Result<ICollection<Message>>> GetUserMessagesAsync(long?
userWorkNodeId, int count = 1, int offset = 0)
    {
        ArgumentNullException.ThrowIfNull(userWorkNodeId);

        if (count < 1)
        {
            throw new ArgumentException(null, nameof(count));
        }

        if (offset < 0)
```

```
        {
            throw new ArgumentException(null, nameof(offset));
        }

        try
        {
            return Result<ICollection<Message>>.FromSuccess(
                await _dbContext.Messages
                    .Include(x => x.Contents)
                    .Where(x => x.UserWorkNodeId == userWorkNodeId)
                    .OrderByDescending(x => x.DateTime)
                    .Skip(offset)
                    .Take(count)
                    .ToListAsync());
        }
        catch (Exception e)
        {
            return
Result<ICollection<Message>>.FromFail((int)ResultStatus.ProcessingError,
e.Message);
        }
    }
}
```

WorkNodeStore.cs

```
using Common.Models;
using Microsoft.EntityFrameworkCore;
using MrMonitor.WorkerService.Enums;
using MrMonitor.WorkerService.Models.Db;
using MrMonitor.WorkerService.Models.Rdb;
using Redis.OM;
using System.Linq;
using System.Text.Json;

namespace MrMonitor.WorkerService.Stores;

public class WorkNodeStore : IWorkNodeStore
{
    private const string WorkNodeNotFoundMessage = "Work node not found.";
    private const int MaxWorkNodeStateBatchSize = 100;

    private readonly WorkerDbContext _dbContext;
    private readonly RedisConnectionProvider _redisConnectionProvider;

    public WorkNodeStore(WorkerDbContext dbContext, RedisConnectionProvider
redisConnectionProvider)
    {
        _dbContext = dbContext ?? throw new
ArgumentNullException(nameof(dbContext));
        _redisConnectionProvider = redisConnectionProvider ?? throw new
ArgumentNullException(nameof(redisConnectionProvider));
    }

    public async Task<Result> CreateWorkNodeAsync(string name, long? creatorId)
    {
        ArgumentException.ThrowIfNullOrEmpty(name);
        ArgumentNullException.ThrowIfNull(creatorId);

        try
        {
            var workNode = new WorkNode
            {
                Name = name,
                Token = GenerateWorkNodeToken(),
                UserWorkNodes = new []
                {
                    new UserWorkNode
                    {
                        UserId = creatorId.Value,
                        Permission = WorkNodePermission.Creator
                    }
                }
```

```
            };

            await _dbContext.WorkNodes.AddAsync(workNode);
            await _dbContext.SaveChangesAsync();

            return Result.FromSuccess();
        }
        catch (Exception e)
        {
            return Result.FromFail((int)ResultStatus.ProcessingError,
e.Message);
        }
    }

    public async Task<Result> ChangeWorkNodeNameAsync(string name, long?
workNodeId)
    {
        ArgumentException.ThrowIfNullOrEmpty(name);
        ArgumentNullException.ThrowIfNull(workNodeId);

        try
        {
            var findWorkNodeResult = await FindWorkNodeAsync(workNodeId);

            if (!findWorkNodeResult.IsSuccess)
            {
                return Result.FromFail(findWorkNodeResult.Status.Value,
findWorkNodeResult.Message);
            }

            findWorkNodeResult.Entity.Name = name;

            await _dbContext.SaveChangesAsync();

            return Result.FromSuccess();
        }
        catch (Exception e)
        {
            return Result.FromFail((int)ResultStatus.ProcessingError,
e.Message);
        }
    }

    public async Task<Result<string>> GetWorkNodeTokenAsync(long? workNodeId)
    {
        ArgumentNullException.ThrowIfNull(workNodeId);

        try
        {
            var findWorkNodeResult = await FindWorkNodeAsync(workNodeId);
```

```
            if (!findWorkNodeResult.IsSuccess)
            {
                return Result<string>.FromFail(findWorkNodeResult.Status.Value,
findWorkNodeResult.Message);
            }

            return Result<string>.FromSuccess(findWorkNodeResult.Entity.Token);
        }
        catch (Exception e)
        {
            return Result<string>.FromFail((int)ResultStatus.ProcessingError,
e.Message);
        }
    }

    public async Task<Result> ResetWorkNodeTokenAsync(long? workNodeId)
    {
        ArgumentNullException.ThrowIfNull(workNodeId);

        try
        {
            var findWorkNodeResult = await FindWorkNodeAsync(workNodeId);

            if (!findWorkNodeResult.IsSuccess)
            {
                return Result.FromFail(findWorkNodeResult.Status.Value,
findWorkNodeResult.Message);
            }

            findWorkNodeResult.Entity.Token = GenerateWorkNodeToken();

            await _dbContext.SaveChangesAsync();

            return Result.FromSuccess();
        }
        catch (Exception e)
        {
            return Result.FromFail((int)ResultStatus.ProcessingError,
e.Message);
        }
    }

    public async Task<Result> RemoveWorkNodeAsync(long? workNodeId)
    {
        ArgumentNullException.ThrowIfNull(workNodeId);

        try
        {
            var findWorkNodeResult = await FindWorkNodeAsync(workNodeId);

            if (!findWorkNodeResult.IsSuccess)
```

```
            {
                return Result.FromFail(findWorkNodeResult.Status.Value,
findWorkNodeResult.Message);
            }

            _dbContext.WorkNodes.Remove(findWorkNodeResult.Entity);

            await _dbContext.SaveChangesAsync();

            return Result.FromSuccess();
        }
        catch (Exception e)
        {
            return Result.FromFail((int)ResultStatus.ProcessingError,
e.Message);
        }
    }

    public async Task<Result<WorkNode>> FindWorkNodeAsync(long? workNodeId)
    {
        ArgumentNullException.ThrowIfNull(workNodeId);

        try
        {
            var workNode = await _dbContext.WorkNodes
                .FirstOrDefaultAsync(x => x.Id == workNodeId);

            if (workNode == null)
            {
                return
Result<WorkNode>.FromFail((int)ResultStatus.WorkNodeNotFound,
WorkNodeNotFoundMessage);
            }

            return Result<WorkNode>.FromSuccess(workNode);
        }
        catch (Exception e)
        {
            return Result<WorkNode>.FromFail((int)ResultStatus.ProcessingError,
e.Message);
        }
    }

    public async Task<Result<WorkNode>> FindWorkNodeAsync(string token)
    {
        ArgumentException.ThrowIfNullOrEmpty(token);

        try
        {
            var workNode = await _dbContext.WorkNodes
                .FirstOrDefaultAsync(x => x.Token == token);
```

```
            if (workNode == null)
            {
                return
Result<WorkNode>.FromFail((int)ResultStatus.WorkNodeNotFound,
WorkNodeNotFoundMessage);
            }

            return Result<WorkNode>.FromSuccess(workNode);
        }
        catch (Exception e)
        {
            return Result<WorkNode>.FromFail((int)ResultStatus.ProcessingError,
e.Message);
        }
    }

    public async Task<Result<UserWorkNode>> GetUserWorkNodeAsync(long? userId,
long workNodeId)
    {
        ArgumentNullException.ThrowIfNull(userId);
        ArgumentNullException.ThrowIfNull(workNodeId);

        try
        {
            var workNode = await _dbContext.UserWorkNodes.FirstOrDefaultAsync(x
=>
                x.UserId == userId && x.WorkNodeId == workNodeId);

            if (workNode == null)
            {
                return
Result<UserWorkNode>.FromFail((int)ResultStatus.WorkNodeNotFound,
WorkNodeNotFoundMessage);
            }

            return Result<UserWorkNode>.FromSuccess(workNode);
        }
        catch (Exception e)
        {
            return
Result<UserWorkNode>.FromFail((int)ResultStatus.ProcessingError, e.Message);
        }
    }

    public async Task<Result<ICollection<UserWorkNode>>>
GetUserWorkNodesAsync(long? userId)
    {
        ArgumentNullException.ThrowIfNull(userId);

        try
```

```
        {
            return Result<ICollection<UserWorkNode>>.FromSuccess(
                await _dbContext.UserWorkNodes
                    .Include(x => x.WorkNode)
                    .Where(x => x.UserId == userId)
                    .ToListAsync());
        }
        catch (Exception e)
        {
            return
Result<ICollection<UserWorkNode>>.FromFail((int)ResultStatus.ProcessingError,
e.Message);
        }
    }

    public async Task<Result<ICollection<long>>> GetWorkNodeUserIdsAsync(long?
workNodeId)
    {
        ArgumentNullException.ThrowIfNull(workNodeId);

        try
        {
            return Result<ICollection<long>>.FromSuccess(
                await _dbContext.UserWorkNodes
                    .Where(x => x.WorkNodeId == workNodeId)
                    .Select(x => x.UserId)
                    .ToListAsync());
        }
        catch (Exception e)
        {
            return
Result<ICollection<long>>.FromFail((int)ResultStatus.ProcessingError,
e.Message);
        }
    }

    public async Task<Result> StoreWorkNodeStateDataAsync(long? workNodeId,
string key, ICollection<WorkNodeState> states)
    {
        ArgumentNullException.ThrowIfNull(workNodeId);
        ArgumentException.ThrowIfNullOrEmpty(key);
        ArgumentNullException.ThrowIfNull(states);

        if (!states.Any())
        {
            return Result.FromSuccess();
        }

        try
        {
            var findWorkNodeResult = await FindWorkNodeAsync(workNodeId);
```

```
            if (!findWorkNodeResult.IsSuccess)
            {
                return Result.FromFail(findWorkNodeResult.Status.Value,
findWorkNodeResult.Message);
            }

            var nonCompleteBatch = await _dbContext.WorkNodeStateData
                .Where(x => x.WorkNodeId == workNodeId && x.EntriesCount <
MaxWorkNodeStateBatchSize)
                .OrderByDescending(x => x.CreatedDateTime)
                .FirstOrDefaultAsync();

            var sortedStackStates = new
Stack<WorkNodeState>(states.OrderByDescending(x => x.DateTime));

            if (nonCompleteBatch != null)
            {
                var nonCompleteBatchData = JsonSerializer
                    .Deserialize<List<WorkNodeState>>(nonCompleteBatch.Data);

                var leftCount = MaxWorkNodeStateBatchSize -
nonCompleteBatchData.Count;

                for (int i = 0; i < leftCount && sortedStackStates.Count > 0;
i++)
                {
                    nonCompleteBatchData.Add(sortedStackStates.Pop());
                }

                nonCompleteBatch.EntriesCount = nonCompleteBatchData.Count;
                nonCompleteBatch.Data = JsonSerializer.Serialize(
                    nonCompleteBatchData.OrderByDescending(x => x.DateTime));
            }

            while (true)
            {
                if (!sortedStackStates.Any())
                {
                    await _dbContext.SaveChangesAsync();

                    return Result.FromSuccess();
                }

                var batchData = new List<WorkNodeState>();

                for (int i = 0; i < MaxWorkNodeStateBatchSize &&
sortedStackStates.Count > 0; i++)
                {
                    batchData.Add(sortedStackStates.Pop());
                }
```

```
                await _dbContext.WorkNodeStateData.AddAsync(new
WorkNodeStateData
                {
                    WorkNodeId = workNodeId.Value,
                    Key = key,
                    EntriesCount = batchData.Count(),
                    Data = JsonSerializer.Serialize(
                        batchData.OrderByDescending(x => x.DateTime)),
                    CreatedDateTime = DateTime.UtcNow
                });
            }
        }
        catch (Exception e)
        {
            return Result.FromFail((int)ResultStatus.ProcessingError,
e.Message);
        }
    }

    public async Task<Result<ICollection<WorkNodeState>>>
GetWorkNodeStateDataAsync(long? workNodeId, string key, int count, int offset)
    {
        ArgumentNullException.ThrowIfNull(workNodeId);
        ArgumentException.ThrowIfNullOrEmpty(key);

        if (count < 1)
        {
            throw new ArgumentException(null, nameof(count));
        }

        if (offset < 0)
        {
            throw new ArgumentException(null, nameof(offset));
        }

        try
        {
            var redisCollection =
_redisConnectionProvider.RedisCollection<WorkNodeState>();

            var redisWorkNodeStatesCount = await redisCollection
                .CountAsync(x => x.WorkNodeId == workNodeId);

            var workNodeStates = (List<WorkNodeState>)await redisCollection
                .Where(x => x.WorkNodeId == workNodeId && x.Key == key)
                .OrderByDescending(x => x.DateTime)
                .Skip(offset)
                .Take(count)
                .ToListAsync();
```

```
            if (count <= redisWorkNodeStatesCount - offset)
            {
                return
Result<ICollection<WorkNodeState>>.FromSuccess(workNodeStates);
            }

            var lastWorkNodeStatesBatch = await _dbContext.WorkNodeStateData
                .Where(x => x.WorkNodeId == workNodeId && x.Key == key)
                .OrderByDescending(x => x.CreatedDateTime)
                .FirstOrDefaultAsync();

            if (lastWorkNodeStatesBatch == null)
            {
                return
Result<ICollection<WorkNodeState>>.FromSuccess(workNodeStates);
            }

            var dbOffset = offset > redisWorkNodeStatesCount
               ? offset - redisWorkNodeStatesCount
               : 0;

            var dbCount = count - workNodeStates.Count;

            var offsetHistory = 0;

            if (dbOffset >= lastWorkNodeStatesBatch.EntriesCount)
            {
                dbOffset -= lastWorkNodeStatesBatch.EntriesCount;
                offsetHistory += dbOffset / MaxWorkNodeStateBatchSize;
                dbOffset -= offsetHistory * MaxWorkNodeStateBatchSize;
                offsetHistory++;
            }

            var historyCount = 0;

            if (offsetHistory == 0)
            {
                historyCount = 1 + (int)Math.Ceiling(
                    (dbCount - (lastWorkNodeStatesBatch.EntriesCount -
dbOffset)) /
                    (double)MaxWorkNodeStateBatchSize);
            }
            else
            {
                historyCount = (int)Math.Ceiling(
                    (dbCount + dbOffset) /
                    (double)MaxWorkNodeStateBatchSize);
            }

            var workNodeStatesHistory = await _dbContext.WorkNodeStateData
                .Where(x => x.WorkNodeId == workNodeId && x.Key == key)
```

```
                .OrderByDescending(x => x.CreatedDateTime)
                .Skip(offsetHistory)
                .Take(historyCount)
                .ToListAsync();

            foreach (var historyData in workNodeStatesHistory)
            {
                workNodeStates.AddRange(JsonSerializer
                    .Deserialize<ICollection<WorkNodeState>>(historyData.Data))
;
            }

            if (!workNodeStates.Any())
            {
                Result<ICollection<WorkNodeState>>.FromSuccess(workNodeStates);
            }

            return
Result<ICollection<WorkNodeState>>.FromSuccess(workNodeStates
                .OrderByDescending(x => x.DateTime)
                .Skip(dbOffset)
                .Take(dbCount)
                .ToList());
        }
        catch (Exception e)
        {
            return
Result<ICollection<WorkNodeState>>.FromFail((int)ResultStatus.ProcessingError,
e.Message);
        }
    }

    private string GenerateWorkNodeToken() =>
Guid.NewGuid().ToString().Replace("-", string.Empty);
}
```

WorkerController.cs

```
using System.Text;
using System.Security.Claims;
using System.IdentityModel.Tokens.Jwt;
using System.ComponentModel.DataAnnotations;
using MrMonitor.Common.Options;
using MrMonitor.WorkerService.Enums;
using MrMonitor.WorkerService.Models.Requests;
using MrMonitor.WorkerService.Models.Responses;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Authorization;
using Microsoft.IdentityModel.Tokens;
using MrMonitor.WorkerService.Stores;
using MrMonitor.WorkerService.Models;

namespace MrMonitor.WorkerService.Controllers;

[ApiController]
[Route("[controller]")]
public class WorkerController : Controller
{
    private const string BadJwtTokenMessage = "Bad JWT token. Please relogin.";
    private const string ServerErrorMessage = "Server error.";

    private readonly IChatStore _chatStore;
    private readonly IWorkNodeStore _workNodeStore;
    private readonly AuthOptions _authOptions;

    public WorkerController(IChatStore chatStore, IWorkNodeStore workNodeStore,
AuthOptions jwtAuthConfiguration)
    {
        _chatStore = chatStore;
        _workNodeStore = workNodeStore;
        _authOptions = jwtAuthConfiguration;
    }

    [Authorize]
    [HttpPut]
    [Route("createWorkNode/{workNodeName}")]
    public async Task<IActionResult> CreateWorkNode(
        [MinLength(3), MaxLength(50)] string workNodeName)
    {
        var userId = GetUserIdFromClaims();

        if (!userId.HasValue)
        {
            return BadRequest(BadJwtTokenMessage);
        }
```

```
        var createWorkNodeResult = await
_workNodeStore.CreateWorkNodeAsync(workNodeName, userId);

        if (createWorkNodeResult.IsSuccess)
        {
            return Ok();
        }

        return StatusCode(StatusCodes.Status500InternalServerError);
    }

    [Authorize]
    [HttpPut]
    [Route("updateWorkNodeInfo")]
    public async Task<IActionResult> UpdateWorkNodeInfo(
        [FromBody, Required] UpdateWorkNodeInfoRequest request)
    {
        var userId = GetUserIdFromClaims();

        if (!userId.HasValue)
        {
            return BadRequest(BadJwtTokenMessage);
        }

        var updateWorkNodeInfoResult = await _workNodeStore
            .ChangeWorkNodeNameAsync(request.WorkNodeName, request.WorkNodeId);

        if (updateWorkNodeInfoResult.IsSuccess)
        {
            return Ok();
        }

        return updateWorkNodeInfoResult.Status ==
(int)ResultStatus.WorkNodeNotFound
            ? NotFound(updateWorkNodeInfoResult.Message)
            : StatusCode(StatusCodes.Status500InternalServerError);
    }

    [Authorize]
    [HttpGet]
    [Route("getWorkNodeToken/{workNodeId}")]
    public async Task<IActionResult> GetWorkNodeToken(long workNodeId)
    {
        var userId = GetUserIdFromClaims();

        if (!userId.HasValue)
        {
            return BadRequest(BadJwtTokenMessage);
        }
```

```
        var getWorkNodeTokenResult = await
_workNodeStore.GetWorkNodeTokenAsync(workNodeId);

        if (getWorkNodeTokenResult.IsSuccess)
        {
            return Ok(getWorkNodeTokenResult.Entity);
        }

        return getWorkNodeTokenResult.Status ==
(int)ResultStatus.WorkNodeNotFound
            ? NotFound(getWorkNodeTokenResult.Message)
            : StatusCode(StatusCodes.Status500InternalServerError);
    }

    [Authorize]
    [HttpPost]
    [Route("resetWorkNodeToken/{workNodeId}")]
    public async Task<IActionResult> ResetWorkNodeToken(long workNodeId)
    {
        var userId = GetUserIdFromClaims();

        if (!userId.HasValue)
        {
            return BadRequest(BadJwtTokenMessage);
        }

        var resetWorkNodeTokenResult = await
_workNodeStore.ResetWorkNodeTokenAsync(workNodeId);

        if (resetWorkNodeTokenResult.IsSuccess)
        {
            return Ok();
        }

        return resetWorkNodeTokenResult.Status ==
(int)ResultStatus.WorkNodeNotFound
            ? NotFound(resetWorkNodeTokenResult.Message)
            : StatusCode(StatusCodes.Status500InternalServerError);
    }

    [Authorize]
    [HttpDelete]
    [Route("deleteWorkNode/{workNodeId}")]
    public async Task<IActionResult> DeleteWorkNode(long workNodeId)
    {
        var userId = GetUserIdFromClaims();

        if (!userId.HasValue)
        {
            return BadRequest(BadJwtTokenMessage);
        }
```

```
        var removeWorkNodeResult = await
_workNodeStore.RemoveWorkNodeAsync(workNodeId);

        if (removeWorkNodeResult.IsSuccess)
        {
            return Ok();
        }

        return removeWorkNodeResult.Status ==
(int)ResultStatus.WorkNodeNotFound
            ? NotFound(removeWorkNodeResult.Message)
            : StatusCode(StatusCodes.Status500InternalServerError);
    }

    [Authorize]
    [HttpGet]
    [Route("getUserWorkNodes")]
    public async Task<IActionResult> GetUserWorkNodes()
    {
        var userId = GetUserIdFromClaims();

        if (!userId.HasValue)
        {
            return BadRequest(BadJwtTokenMessage);
        }

        var getUserWorkNodesResult = await
_workNodeStore.GetUserWorkNodesAsync(userId.Value);

        if (getUserWorkNodesResult.IsSuccess)
        {
            return Ok(new GetUserWorkNodesResponse
            {
                Values = getUserWorkNodesResult.Entity
                    .Select(x => new WorkNodeInfo
                    {
                        WorkNodeId = x.WorkNode.Id,
                        WorkNodeName = x.WorkNode.Name
                    })
                    .ToList()
            });
        }

        return StatusCode(StatusCodes.Status500InternalServerError);
    }

    [HttpGet]
    [Route("getWorkNodeAccessToken/{token}")]
    public async Task<IActionResult> GetWorkNodeAccessToken(string token)
    {
```

```
        var findWorkNodeResult = await _workNodeStore.FindWorkNodeAsync(token);

        if (!findWorkNodeResult.IsSuccess)
        {
            return findWorkNodeResult.Status ==
(int)ResultStatus.WorkNodeNotFound
                ? NotFound(findWorkNodeResult.Message)
                : StatusCode(StatusCodes.Status500InternalServerError,
ServerErrorMessage);
        }

        var tokenHandler = new JwtSecurityTokenHandler();

        var accessToken = tokenHandler.CreateToken(new SecurityTokenDescriptor
        {
            Subject = new ClaimsIdentity(new[]
            {
                new Claim("workNodeId",
findWorkNodeResult.Entity.Id.ToString())
            }),
            Expires = DateTime.UtcNow.AddDays(14),
            Issuer = _authOptions.Issuer,
            Audience = _authOptions.Audience,
            SigningCredentials = new SigningCredentials(
                new
SymmetricSecurityKey(Encoding.UTF8.GetBytes(_authOptions.SecretKey)),
                SecurityAlgorithms.HmacSha512Signature)
        });

        return Ok(tokenHandler.WriteToken(accessToken));
    }

    [Authorize]
    [HttpGet]
    [Route("getUserMessages/{workNodeId}")]
    public async Task<IActionResult> GetUserMessages(
        long? workNodeId,
        [FromQuery, Range(1, int.MaxValue)] int? count,
        [FromQuery, Range(0, int.MaxValue)] int? offset)
    {
        var userId = GetUserIdFromClaims();

        if (!userId.HasValue)
        {
            return BadRequest(BadJwtTokenMessage);
        }

        var getUserWorkNodeResult = await
_workNodeStore.GetUserWorkNodeAsync(userId, workNodeId.Value);

        if (!getUserWorkNodeResult.IsSuccess)
```

```
        {
            return getUserWorkNodeResult.Status ==
(int)ResultStatus.WorkNodeNotFound
                ? NotFound(getUserWorkNodeResult.Message)
                : StatusCode(StatusCodes.Status500InternalServerError);
        }

        var getUserMessagesResult = await _chatStore.GetUserMessagesAsync(
            getUserWorkNodeResult.Entity.Id,
            count.Value,
            offset.Value);

        if (getUserMessagesResult.IsSuccess)
        {
            return Ok(new GetUserWorkNodeMessagesResponse
            {
                Values = getUserMessagesResult.Entity
                    .Select(x => new Message
                    {
                        Id = x.Id,
                        SenderType = x.SenderType,
                        DateTime = new DateTime(
                            x.DateTime.Year,
                            x.DateTime.Month,
                            x.DateTime.Day,
                            x.DateTime.Hour,
                            x.DateTime.Minute,
                            x.DateTime.Second,
                            x.DateTime.Microsecond,
                            DateTimeKind.Utc),
                        Contents = x.Contents
                            .OrderBy(x => x.Id)
                            .Select(x => new MessageContent
                            {
                                Type = x.Type,
                                Data = x.Data
                            })
                            .ToList()
                    })
                    .OrderByDescending(x => x.Id)
                    .ToList()
            });
        }

        return StatusCode(StatusCodes.Status500InternalServerError);
    }

    [Authorize]
    [HttpGet]
    [Route("getWorkNodeStates/{workNodeId}/{key}")]
    public async Task<IActionResult> GetWorkNodeStates(
```

```
        long? workNodeId,
        string key,
        [FromQuery, Range(1, int.MaxValue)] int? count,
        [FromQuery, Range(0, int.MaxValue)] int? offset)
    {
        var userId = GetUserIdFromClaims();

        if (!userId.HasValue)
        {
            return BadRequest(BadJwtTokenMessage);
        }

        var getUserWorkNodeResult = await
_workNodeStore.GetUserWorkNodeAsync(userId, workNodeId.Value);

        if (!getUserWorkNodeResult.IsSuccess)
        {
            return getUserWorkNodeResult.Status ==
(int)ResultStatus.WorkNodeNotFound
                ? NotFound(getUserWorkNodeResult.Message)
                : StatusCode(StatusCodes.Status500InternalServerError);
        }

        var getWorkNodeStateDataResult = await
_workNodeStore.GetWorkNodeStateDataAsync(
            workNodeId.Value,
            key,
            count.Value,
            offset.Value);

        if (getWorkNodeStateDataResult.IsSuccess)
        {
            return Ok(new GetWorkNodeStatesResponse
            {
                Values = getWorkNodeStateDataResult.Entity
                    .Select(x => new WorkNodeState
                    {
                        Id = x.Id,
                        Key = x.Key,
                        Value = x.Value,
                        DateTime = x.DateTime
                    })
                    .ToList()
            });
        }

        return StatusCode(StatusCodes.Status500InternalServerError);
    }

    private long? GetUserIdFromClaims()
    {
```

```
        var userIdClaim = User.Claims.FirstOrDefault(x => x.Type == "userId");
        long userId;

        if (userIdClaim == null || !long.TryParse(userIdClaim.Value, out
userId))
        {
            return null;
        }

        return long.Parse(userIdClaim.Value);
    }
}
```

ChatHub.cs

```
using Microsoft.AspNetCore.SignalR;
using Microsoft.AspNetCore.Authorization;
using MrMonitor.WorkerService.Stores;
using MrMonitor.WorkerService.Enums;
using MrMonitor.WorkerService.Models;
using Redis.OM.Searching;
using Redis.OM;

namespace MrMonitor.WorkerService.Hubs;

[Authorize]
public class ChatHub : Hub
{
    private static List<HubConnectionMap> _connectionMaps = new
List<HubConnectionMap>();

    private readonly IChatStore _chatStore;

    private readonly IWorkNodeStore _workNodeStore;

    private readonly IRedisCollection<Models.Rdb.WorkNodeState> _states;

    public ChatHub(IChatStore chatStore, IWorkNodeStore workNodeStore,
RedisConnectionProvider redisConnectionProvider)
    {
        _chatStore = chatStore ?? throw new
ArgumentNullException(nameof(chatStore));
        _workNodeStore = workNodeStore ?? throw new
ArgumentNullException(nameof(WorkNodeStore));

        ArgumentNullException.ThrowIfNull(redisConnectionProvider);

        _states =
redisConnectionProvider.RedisCollection<Models.Rdb.WorkNodeState>();
    }

    public override Task OnConnectedAsync()
    {
        var connectedClientId = GetIdFromClaims("userId");
        var connectedClientType = HubConnectionOwnerType.User;

        if (connectedClientId == null)
        {
            connectedClientId = GetIdFromClaims("workNodeId");
            connectedClientType = HubConnectionOwnerType.WorkNode;
        }
```

```
        if (connectedClientId == null)
        {
            Context.Abort();

            return base.OnConnectedAsync();
        }

        var map = _connectionMaps.FirstOrDefault(x =>
            x.OwnerId == connectedClientId);

        if (map != null)
        {
            map.Connections.Add(Context.ConnectionId);
        }
        else
        {
            _connectionMaps.Add(new HubConnectionMap
            {
                OwnerId = connectedClientId.Value,
                OwnerType = connectedClientType,
                Connections = new List<string> { Context.ConnectionId}
            });
        }

        return base.OnConnectedAsync();
    }

    public override Task OnDisconnectedAsync(Exception? exception)
    {
        var map = _connectionMaps.FirstOrDefault(x =>
            x.Connections.Contains(Context.ConnectionId));

        if (map != null)
        {
            map.Connections.Remove(Context.ConnectionId);

            if (map.Connections.Count == 0)
            {
                _connectionMaps.Remove(map);
            }
        }

        return base.OnDisconnectedAsync(exception);
    }

    [Authorize]
    public async Task OnUpdate(Update update)
    {
        update.Id = Guid.NewGuid().ToString();

        var senderId = GetIdFromClaims("userId");
```

```
        if (senderId != null)
        {
            await ProcessUserUpdate(senderId.Value, update);
            return;
        }

        senderId = GetIdFromClaims("workNodeId");

        if (senderId != null)
        {
            await ProcessWorkNodeUpdate(senderId.Value, update);
        }
    }

    private async Task ProcessUserUpdate(long userId, Update update)
    {
        var userConnectionMap = _connectionMaps.FirstOrDefault(x =>
            x.OwnerId == userId && x.OwnerType == HubConnectionOwnerType.User);

        if (userConnectionMap == null)
        {
            return;
        }

        var workNodeId = update.WorkNodeId;
        var getUserWorkNodeResult = await
_workNodeStore.GetUserWorkNodeAsync(userId, workNodeId);

        if (!getUserWorkNodeResult.IsSuccess)
        {
            return;
        }

        var now = DateTime.UtcNow;
        var dbMessage = new Models.Db.Message
        {
            SenderType = SenderType.User,
            UserWorkNodeId = getUserWorkNodeResult.Entity.Id,
            Contents = update.Message.Contents
                .Select(c => new Models.Db.MessageContent
                {
                    Type = c.Type,
                    Data = c.Data
                })
                .ToList(),
            DateTime = new DateTime(
                    now.Year,
                    now.Month,
                    now.Day,
                    now.Hour,
                    now.Minute,
```

```
                    0,
                    DateTimeKind.Utc)
        };

        var addMessageResult = await _chatStore.AddMessageAsync(dbMessage);

        if (!addMessageResult.IsSuccess)
        {
            return;
        }

        update.WorkNodeId = workNodeId;
        update.UserId = userId;
        update.SenderType = SenderType.User;
        update.Message.Id = dbMessage.Id;
        update.Message.SenderType = dbMessage.SenderType;
        update.Message.DateTime = dbMessage.DateTime;

        var workNodeConnectionsMap = _connectionMaps.FirstOrDefault(x =>
            x.OwnerId == workNodeId && x.OwnerType ==
HubConnectionOwnerType.WorkNode);

        if (workNodeConnectionsMap != null)
        {
            await
Clients.Clients(workNodeConnectionsMap.Connections).SendAsync("OnUpdate",
update);
        }

        await
Clients.Clients(userConnectionMap.Connections).SendAsync("OnUpdate", update);
    }

    private async Task ProcessWorkNodeUpdate(long workNodeId, Update update)
    {
        var workNodeConnectionMap = _connectionMaps.FirstOrDefault(x =>
x.OwnerId == workNodeId);

        if (workNodeConnectionMap == null)
        {
            return;
        }

        update.WorkNodeId = workNodeId;

        if (update.Type == UpdateType.Message)
        {
            var userId = update.UserId;

            var getUserWorkNodeResult = await
_workNodeStore.GetUserWorkNodeAsync(userId, workNodeId);
```

```
            if (!getUserWorkNodeResult.IsSuccess)
            {
                return;
            }

            var dbMessage = new Models.Db.Message
            {
                SenderType = SenderType.WorkNode,
                UserWorkNodeId = getUserWorkNodeResult.Entity.Id,
                Contents = update.Message.Contents
                .Select(c => new Models.Db.MessageContent
                {
                    Type = c.Type,
                    Data = c.Data
                })
                .ToList(),
                DateTime = DateTime.UtcNow
            };

            var addMessageResult = await _chatStore.AddMessageAsync(dbMessage);

            if (!addMessageResult.IsSuccess)
            {
                return;
            }

            update.UserId = userId;
            update.SenderType = SenderType.User;
            update.Message.Id = dbMessage.Id;
            update.Message.SenderType = dbMessage.SenderType;
            update.Message.DateTime = dbMessage.DateTime;

            var userConnectionsMap = _connectionMaps
                .FirstOrDefault(x => x.OwnerId == userId);

            if (userConnectionsMap != null)
            {
                await
Clients.Clients(userConnectionsMap.Connections).SendAsync("OnUpdate", update);
            }
        }

        if (update.Type == UpdateType.Frame)
        {
            var getWorkNodeUserIds = await
_workNodeStore.GetWorkNodeUserIdsAsync(workNodeId);

            if (!getWorkNodeUserIds.IsSuccess)
            {
                return;
```

```
            }

            await _states.InsertAsync(update.Frame.States
                .Select(state => new Models.Rdb.WorkNodeState
                {
                    Id = Guid.NewGuid().ToString().Replace("-", ""),
                    Key = state.Key,
                    Value = state.Value,
                    WorkNodeId = workNodeId,
                    DateTime = state.DateTime,
                })
                .ToArray());

            var userIds = getWorkNodeUserIds.Entity;

            var targetConnections = new List<string>();

            foreach(var map in _connectionMaps)
            {
                if (map.OwnerType == HubConnectionOwnerType.WorkNode ||
!userIds.Contains(map.OwnerId))
                {
                    continue;
                }

                targetConnections.AddRange(map.Connections);
            }

            if (targetConnections.Any())
            {
                await Clients.Clients(targetConnections).SendAsync("OnUpdate",
update);
            }
        }
    }

    private long? GetIdFromClaims(string claimType)
    {
        var claim = Context.User.Claims
            .FirstOrDefault(x => x.Type == claimType);

        long id;

        if (claim == null || !long.TryParse(claim.Value, out id))
        {
            return null;
        }

        return long.Parse(claim.Value);
    }
}
```

MainView.js

```
import { useEffect, useRef, useState } from 'react';
import { HubConnectionBuilder } from '@microsoft/signalr'
import Layout from '../../common/layout/Layout';
import SideBar from '../../common/side-bar/SideBar';
import AccountPanel from './components/account-panel/AccountPanel';
import AccountSettingsDialog from './components/dialogs/account-settings-
dialog/AccountSettingsDialogs';
import WorkNodeListPanel from './components/work-node-list-
panel/WorkNodeListPanel';
import CreateWorkNodeDialog from './components/dialogs/create-work-node-
dialog/CreateWorkNodeDialog';
import ControlPanel from './components/control-panel/ControlPanel';
import WorkNodeSettingsDialog from './components/dialogs/work-node-settings-
dialog/WorkNodeSettingsDialog';

function MainView() {

    const hubConnection = useRef(null);
    const lastUpdate = useRef(null);
    const messagesPulling = useRef(false);
    const reportData = useRef(new Set());

    const [viewState, setViewState] = useState(
        {
            dialog: null,
            user: {
                userId: 0,
                firstname: "",
                lastname: "",
                username: ""
            },
            workNodes: [],
            selectedWorkNode: null,
            messagesBatch: {
                values: [],
                batchComplete: undefined
            },
            statesBatch: []
        }
    )
    const [update, setUpdate] = useState(null);

    const setState = (transform) => setViewState(prev => {
        const newState = {
            dialog: prev.dialog,
            user: prev.user,
            workNodes: prev.workNodes,
```

```
            selectedWorkNode: prev.selectedWorkNode,
            messagesBatch: prev.messagesBatch,
            statesBatch: prev.statesBatch
        };

        transform(newState);

        return newState;
    });
    const setDialog = (dialog) => setState(state => state.dialog = dialog);
    const setUser = (user) => setState(state => state.user = user);
    const setMessages = (messages) => setState(state =>
state.messagesBatch.values = messages);

    const connectToHub = () => {
        try {
            hubConnection.current = new HubConnectionBuilder()
                .withUrl('http://localhost:5002/hubs/chat', {
                    accessTokenFactory: () => localStorage.getItem('jwt')
                })
                .build()

            hubConnection.current.on("onUpdate", update => setUpdate(update));
            hubConnection.current.start();
        } catch (exception) {
            alert(exception);
        }
    };

    const get = (url, responseHandler) =>
        fetch(url, {
            method: 'GET',
            headers: {
                'Authorization': `Bearer ${localStorage.getItem("jwt")}`
            }
        })
        .then(response => responseHandler(response))
        .catch(ex => alert(ex));

    const getMe = (onSuccess) =>
        get('http://localhost:5001/auth/getMe', response => {

            if (response.ok) {
                response.json().then(data => onSuccess(data));

                return;
            }

            response.text().then(text => alert(text));
        });
```

```
    const getWorkNodes = (onSuccess) =>
        get('http://localhost:5002/worker/getUserWorkNodes', response => {

            if (response.ok) {
                response.json().then(data => onSuccess(data));

                return;
            }

            response.text().then(text => alert(text));
        });

    const pullMessages = (workNodeId, count, offset, onSuccess) =>
        get(
            `http://localhost:5002/worker/getUserMessages/${workNodeId}?count=$
{count}&offset=${offset}`,
            response => {

                if (!response.ok) {
                    messagesPulling.current = false;
                    response.text().then(text => alert(text));

                    return;
                }

                response.json().then(data => onSuccess(data));
            }
        );

    const pullStates = (workNodeId, key, count, offset, onSuccess) =>
        get(
            `http://localhost:5002/worker/getWorkNodeStates/${workNodeId}/${key
}?count=${count}&offset=${offset}`,
            response => {

                if (!response.ok) {
                    response.text().then(text => alert(text));

                    return;
                }

                response.json().then(data => onSuccess(data));
            }
        );

    useEffect(() => {
        getMe(result => setUser(result));
        getWorkNodes(result => setState(state => state.workNodes =
result.values))
        connectToHub()
    }, []);
```

```
    useEffect(() => {

        if (viewState.messagesBatch.length == 0) {
            return;
        }

        const target = new Map();

        viewState.messagesBatch.values.forEach(message => {
            message.contents.forEach(content => {

                if (content.type === 0) {
                    return;
                }

                const key = content.data.key;

                const currentKeyStates = viewState.statesBatch.find(s => s.key
=== key)

                if (currentKeyStates && currentKeyStates.batchComplete) {
                    return;
                }

                const currentKeyStatesValuesCount = currentKeyStates ?
currentKeyStates.values.length : 0;

                const targetValuesCount = content.type == 1 ? 1 : 100;

                if (content.type == 1 && currentKeyStatesValuesCount >=
targetValuesCount) {
                    return;
                }

                if (content.type == 2 && currentKeyStatesValuesCount >=
targetValuesCount + content.data.offsetLeft) {
                    return;
                }

                if (!target.has(content.data.key) ||
target.get(content.data.key) < targetValuesCount) {
                    target.set(content.data.key, {
                        count: targetValuesCount,
                        offset: currentKeyStatesValuesCount
                    });
                }
            });
        });

        const pulledStates = {
```

```
            values: [...viewState.statesBatch]
        };

        const pulledKey = new Set();

        target.forEach((data, key) => {
            pullStates(
                viewState.selectedWorkNode.workNodeId,
                key,
                data.count,
                data.offset,
                result => {
                    const keyStatesIndex = pulledStates.values.findIndex(s =>
s.key == key);

                    if (keyStatesIndex < 0) {
                        pulledStates.values = [...pulledStates.values, {
                            key: key,
                            values: result.values,
                            batchComplete: data.count > result.values.length
                        }]
                    } else {
                        pulledStates.values[keyStatesIndex].values =
[...pulledStates.values[keyStatesIndex].values, ...result.values];
                        pulledStates.values[keyStatesIndex].batchComplete =
data.count > result.values.length;
                    }

                    pulledKey.add(key);

                    if (pulledKey.size === target.size) {
                        setState(state => state.statesBatch =
pulledStates.values);
                    }
                });
        });

    }, [viewState.messagesBatch.values]);

    useEffect(() => {

        if (!update) {
            return;
        }

        if (lastUpdate.current === update.id) {
            return;
        }

        if (!viewState.selectedWorkNode) {
            return;
```

```
        }

        if (viewState.selectedWorkNode.workNodeId !== update.workNodeId) {
            return;
        }

        if (update.type == 0) {
            setMessages([
                {
                    id: update.message.id,
                    senderType: update.message.senderType,
                    dateTime: update.message.dateTime,
                    contents: update.message.contents.map(content => {
                        const mappedMessage = {
                            type: content.type,
                            data: JSON.parse(content.data)
                        }

                        if (content.type === 2) {
                            mappedMessage.data.offsetLeft = 0;
                            mappedMessage.data.tickCount = 10;
                        }

                        return mappedMessage;
                    })
                },
                ...viewState.messagesBatch.values
            ]);
        }

        if (update.type == 1) {
            setState(state => {

                reportData.current.add({
                    frame: update.frame.states[0].value,
                    time: new Date()
                });

                if (reportData.current.size === 100) {
                    console.log(reportData.current);
                }

                state.statesBatch = state.statesBatch.map(keyStates => {

                    const newState = update.frame.states.find(state =>
keyStates.key === state.key);

                    if (!newState) {
                        return keyStates;
                    }
```

```
                    return {
                        key: keyStates.key,
                        values: [{
                            value: newState.value,
                            dateTime: newState.dateTime
                        }, ...keyStates.values],
                        batchComplete: keyStates.batchComplete
                    }
                });
                state.messagesBatch.values =
state.messagesBatch.values.map(message => ({
                    id: message.id,
                    senderType: message.senderType,
                    dateTime: message.dateTime,
                    contents: message.contents.map(content => {

                        if (content.type !== 2) {
                            return content;
                        }

                        return {
                            type: content.type,
                            data: {
                                key: content.data.key,
                                offsetLeft: content.data.offsetLeft != 0
                                    ? content.data.offsetLeft + 1
                                    : 0,
                                tickCount: content.data.tickCount
                            }
                        }
                    })
                }));
            });
        }
    }, [update]);

    const onAccountSettingsButtonClick = () =>
        setDialog(
            <AccountSettingsDialog
                user={ viewState.user }
                onEdited={onAccountEdited}
                onClose={onCloseDialog} />
        );

    const onCreateWorkNodeButtonClick = () =>
        setDialog(
            <CreateWorkNodeDialog
                onCreated={onWorkNodeCreated}
                onClose={onCloseDialog} />
        );
```

```
    const onWorkNodeSettingsButtonClick = () =>
        setDialog(
            <WorkNodeSettingsDialog
                workNode={viewState.selectedWorkNode}
                onEdited={onWorkNodeEdited}
                onDeleted={onWorkNodeDeleted}
                onClose={onCloseDialog} />
        );

    const onCloseDialog = () => setDialog(null);

    const onCloseControlPanel = () => {
        messagesPulling.current = false;

        setState(state => {
            state.selectedWorkNode = null;
            state.messagesBatch.values = [];
            state.messagesBatch.batchComplete = false;
            state.statesBatch = [];
        });
    }

    const onAccountEdited = changedUser => setUser(changedUser);

    const onWorkNodeCreated = () =>
        getWorkNodes(
            result => {
                setState(state => {
                    state.workNodes = result.values;
                    state.dialog = null;
                })
            },
            message => alert(message)
        );

    const onWorkNodeEdited = changedWorkNode =>
        setState(state => {
            state.workNodes = state.workNodes.map(workNode => {
                if (workNode.workNodeId === changedWorkNode.workNodeId) {
                    workNode.workNodeName = changedWorkNode.workNodeName;
                }

                return workNode;
            });
            state.selectedWorkNode = changedWorkNode;
        });

    const onWorkNodeDeleted = deletedWorkNode =>
        setState(state => {
            state.dialog = null;
            state.workNodes = state.workNodes.filter(workNode =>
```

```
                workNode.workNodeId !== deletedWorkNode.workNodeId);
            state.selectedWorkNode = null;
        });

    const onWorkNodeSelected = selectedWorkNode => {

        if (viewState.selectedWorkNode && viewState.selectedWorkNode.workNodeId
=== selectedWorkNode.workNodeId) {
            return;
        }

        messagesPulling.current = true;

        const pullCount = 100;
        const pullOffset = viewState.messagesBatch.values.length;

        pullMessages(
            selectedWorkNode.workNodeId,
            pullCount,
            pullOffset,
            result => {
                setState(state => {
                    state.selectedWorkNode = selectedWorkNode;
                    state.messagesBatch.values = result.values.map(message =>
({
                        id: message.id,
                        senderType: message.senderType,
                        dateTime: message.dateTime,
                        contents: message.contents.map(content => {
                            const mappedMessage = {
                                type: content.type,
                                data: JSON.parse(content.data)
                            }

                            if (content.type === 2) {
                                mappedMessage.data.offsetLeft = 0;
                                mappedMessage.data.tickCount = 10;
                            }

                            return mappedMessage;
                        })
                    }));
                    state.messagesBatch.batchComplete = result.values.length <
pullCount;
                });

                messagesPulling.current = false;
            }
        );
    }
```

```
    const onMessageListScroll = e => {
        const maxScroll = e.target.scrollHeight;
        const currentScroll = e.target.scrollTop;

        if (messagesPulling.current || maxScroll - Math.abs(currentScroll) >
600) {
            return;
        }

        messagesPulling.current = true;

        const pullCount = 100;
        const pullOffset = viewState.messagesBatch.values.length;

        pullMessages(
            viewState.selectedWorkNode.workNodeId,
            pullCount,
            pullOffset,
            result => {

                const dublicateMessages = new Set();

                setState(state => {
                    state.messagesBatch.values =  [
                        ...state.messagesBatch.values,
                        ...result.values.map(message => ({
                            id: message.id,
                            senderType: message.senderType,
                            dateTime: message.dateTime,
                            contents: message.contents.map(content => {
                                const mappedMessage = {
                                    type: content.type,
                                    data: JSON.parse(content.data)
                                }

                                if (content.type === 2) {
                                    mappedMessage.data.offsetLeft = 0;
                                    mappedMessage.data.tickCount = 10;
                                }

                                return mappedMessage;
                            })
                        }))
                    ].filter(message => {

                        if (dublicateMessages.has(message.id)) {
                            return false;
                        }

                        dublicateMessages.add(message.id);
```

```
                        return true;
                    });

                    state.messagesBatch.batchComplete = result.values.length <
pullCount;
                });
            }
        );
    }

    const onChartOffset = e => {
        setMessages(viewState.messagesBatch.values.map(message => {

            if (message.id !== e.messageId) {
                return message;
            }

            return {
                id: message.id,
                senderType: message.id,
                dateTime: message.dateTime,
                contents: message.contents.map((content, index) => {

                    if (index !== e.contentIndex) {
                        return content;
                    }

                    return {
                        type: content.type,
                        data: {
                            key: content.data.key,
                            offsetLeft: content.data.offsetLeft -
e.scrollDirection,
                            tickCount: content.data.tickCount
                        }
                    }
                })
            }
        }))
    }

    const onChartTickCountChanged = e => {
        setMessages(viewState.messagesBatch.values.map(message => {

            if (message.id !== e.messageId) {
                return message;
            }

            return {
                id: message.id,
                senderType: message.id,
```

```
                dateTime: message.dateTime,
                contents: message.contents.map((content, index) => {

                    if (index !== e.contentIndex) {
                        return content;
                    }

                    return {
                        type: content.type,
                        data: {
                            key: content.data.key,
                            offsetLeft: content.data.offsetLeft,
                            tickCount: parseInt(e.newTickCount)
                        }
                    }
                })
            }
        }))
    }

    const onSendMessage = message => {
        hubConnection.current.send("onUpdate", {
            workNodeId: viewState.selectedWorkNode.workNodeId,
            userId: viewState.user.userId,
            type: 0,
            senderType: 0,
            message: {
                contents: [
                    {
                        type: 0,
                        data: JSON.stringify({
                            text: message
                        })
                    }
                ]
            }
        });
    }

    return (
        <Layout className="MainView">
            { viewState.dialog }
            <SideBar>
                <AccountPanel
                    user={ viewState.user }
                    onClick={onAccountSettingsButtonClick} />

                <WorkNodeListPanel
                    workNodes={ viewState.workNodes }
                    selectedWorkNode={ viewState.selectedWorkNode }
                    onWorkNodeSelected={onWorkNodeSelected}
```

```
                    onCreateWorkNodeButtonClick={onCreateWorkNodeButtonClick}
/>
            </SideBar>

            <ControlPanel
                workNode={ viewState.selectedWorkNode }
                messages={ viewState.messagesBatch.values }
                states={ viewState.statesBatch }
                onSendMessage={ onSendMessage }
                onMessageListScroll={ onMessageListScroll }
                onChartOffset={ onChartOffset }
                onChartTickCountChanged={ onChartTickCountChanged }
                onSettingButtonClick={ onWorkNodeSettingsButtonClick }
                onClose={ onCloseControlPanel } />

        </Layout>
    );
}

export default MainView;
```

ДОДАТОК В

Демонстраційний графічний матеріал

№	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Дод. відомості
		Текстові документи	
1	ГЮІК.627914.039 ПЗ	Пояснювальна записка	75 с. ф. А4
2	ГЮІК.627914.039	Апробація результатів наукових досліджень	14 с. ф. А4
3	ГЮІК.627914.039.505900	Текст програми	61 с. ф. А4
		Графічні документи	
4		Демонстраційний графічний матеріал	у форматі
			*.ptt
		ГЮІК.627914.039 ВД	

Змін.	Арк.	Номер докум.	Підп.	Дата	Розроблення системи автоматизованого моніторингу технічного стану мобільних роботів на підприємстві Відомість кваліфікаційної роботи	Літера	Арку	А
Розроб.		Кравченко С.В.				Н	1	1
Перевір.		Бронніков А.І.				Кафедра КІТАР ХНУРЕ		
Н.контр.		Стародубцев М.Г.						
Затв.		Невлюдов І.Ш.						