Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно–інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: «Розробка інформаційно-аналітичної підсистеми для самодіагностики в кіберфізичних системах»

Виконав: студент IV курсу, групи СП–43

спеціальності 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

(підпис) Чичота П.Я. (прізвище та ініціали)

Керівник (підпис) Бревус В.М. (прізвище та ініціали)

Нормоконтроль (підпис) Стоянов Ю.М. (прізвище та ініціали)

Завідувач кафедри (підпис) Петрик М.Р. (прізвище та ініціали)

Рецензент (підпис) (прізвище та ініціали)

Тернопіль
2025

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно–інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра програмної інженерії
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Петрик М.Р.
(підпис) (прізвище та ініціали)
«___» _червня_ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня Бакалавр
(назва освітнього ступеня)
за спеціальністю 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)
Студенту Чичоті Павлу Ярославовичу
(прізвище, ім'я, по батькові)
1. Тема роботи «Розробка інформаційно-аналітичної підсистеми для самодіагностики в кіберфізичних системах»

Керівник роботи Бревус Віталій Миколайович, к.т.н., доцент кафедри ПІ
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
Затверджені наказом ректора від «*18*» *квітня* 2025 року № *4/7–335*
2. Термін подання студентом завершеної роботи червня 2025 р.
3. Вихідні дані до роботи Наукові публікації про підсистеми для самодіагностики в кіберфізичних системах, інформаційно-аналітичні системи.

4. Зміст роботи: Вступ.
1 Аналіз предметної області та постановка завдання. 1.1 Визначення умов.
1.2 Визначення завдань. 1.3 Аналіз стану концепції КФС. 1.4 Автоматизація роботи з текстом
1.5 Виявлення збоїв. 1.6 Архітектурні особливості KDT,DSL,ANTLR та REST 1.7 Висновки
До першого розділу 2 Проектна частина. 2.1 Структура системи. 2.2 Клієнтська та серверна частина. 2.3.1 Архітектура управління тестуванням та конфігурацією. 2.3.2 Архітектура тестової системи самодіагностики. 2.4 Висновок до другого розділу. 3 Практична частина. Реалізація тестової системи самодіагностики. 3.1 База даних. 3.2 Бекенд. 3.2.1 Моделі
3.2.2 Граматика. 3.2.3 Контролери. 3.2.4 Маршрути. 3.3 Фронтенд. 3.3.1 Компоненти.
3.3.2 Отримання даних API. 3.3.3 Інтерфейси користувача. 3.4 Валідація. 3.5 Висновок до третього розділу. 4. Безпека життєдіяльності, основи охорони праці. Висновки. Перелік джерел. Додатки
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
1 Титульна сторінка. 2 Тема, Мета, Об'єкт, Предмет дослідження. 3 Завдання дослідження.
4 Актуальність дослідження. 5 Схема мовного процесора. 6 Об'єктне мапування між Node.js та MongoDB, кероване через Mongoose. 7 Реальний DOM та React Virtual DOM.
8 Підхід KDT. 9 Пропонована архітектура для KDT з DSL. 10 Архітектура тестової системи самодіагностики. 11 Архітектура тестової системи самодіагностики, інтегрованої з КФС.
12 Сторінка виконання а також Таблиця результатів виконання. 13 Сторінка звіту.
14 Сторінки керування і конфігурації 15 Висновки. 16 Завершальний.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Охорона праці та Безпека в надзвичайних ситуаціях	Лазарюк В.В., к.т.н., доц. кафедри МТ	02.06.2025	05.06.2025

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Ознайомлення з завданням до кваліфікаційної роботи	01.05.2025	*Виконано*
2.	Підбір та опрацювання літературних джерел по темі	20.04.2025–30.04.2025	*Виконано*
	кваліфікаційної роботи		
3.	Виконання дослідження згідно мети кваліфікаційної	1.05.2025–10.05.2025	*Виконано*
	роботи		
4.	Оформлення розділу «Аналіз предметної області та	11.05.2025–20.05.2025	*Виконано*
	постановка завдання»		
5.	Оформлення розділу «Проектна частина. Аналіз та	21.05.2025–25.05.2025	*Виконано*
	специфікація»		
6.	Оформлення розділу «Практична частина. Реалізація	23.05.2025–01.06.2025	*Виконано*
	тестової системи самодіагностики»		
7.	Виконання завдання до підрозділу «Безпека життєдіяльності»	02.06.2025–05.06.2025	*Виконано*
8.	Виконання завдання до підрозділу «Основи охорони	02.06.2025–05.06.2025	*Виконано*
	праці»		
9.	Оформлення кваліфікаційної роботи	06.06.2025–10.06.2025	*Виконано*
10.	Нормоконтроль	11.06.2025–13.06.2025	*Виконано*
11.	Перевірка на плагіат	16.06.2025	*Виконано*
12.	Попередній захист кваліфікаційної роботи	9.06.2025	*Виконано*
13.	Захист кваліфікаційної роботи		

Студент _____ Чичота П.Я.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Бревус В.М.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Розробка інформаційно-аналітичної підсистеми для самодіагностики в кіберфізичних системах // Кваліфікаційна робота освітнього рівня «Бакалавр» // Чичота Павло Ярославович // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп'ютерно–інформаційних систем і програмної інженерії, кафедра програмної інженерії, група СП–43 // Тернопіль, 2025 // С. 62 , рис. – 13, табл. – 1, кресл. – 16 , додат. – 5 , бібліогр. – 50.

Ключові слова: кіберфізичні системи; самодіагностика; автоматизація тестування; веб- застосунки, інформаційно-аналітична підсистема.

Кваліфікаційна робота присвячена дослідженню розробки підсистеми самодіагностики в кіберфізичних системах для підвищення їх надійності.

У першому розділі описано особливості КФС, розглянуто проблеми тестування, висвітлено методи автоматизації та проаналізовано сучасні підходи до діагностики.

У другому розділі досліджено вимоги до системи, подано архітектуру підсистеми, обґрунтовано вибір технологій та моделювання її компонентів.

У третьому розділі описано реалізацію системи, проаналізовано її основні модулі, проведено тестування працездатності і функціоналу.

У четвертому розділі розглядаються питання забезпечення безпечної експлуатації обладнання.

Об'єкт дослідження: кіберфізичні системи, що потребують автоматизованої самодіагностики.

Предмет дослідження: інформаційно-аналітична підсистема для виявлення та аналізу збоїв у роботі кіберфізичних систем, побудована з використанням методів автоматизованого тестування та мовної інтерпретації сценаріїв.

ABSTRACT

Development of an information and analytical subsystem for self-diagnosis in cyber-physical systems // Qualification work of the educational level «Bachelor» // Chychota Pavlo Yaroslavovych // Ternopil Ivan Pulyu National Technical University, Computer and Information Systems and Software Engineering Faculty, Software Engineering Department, group SP–43 // Ternopil, 2025 // P. 67, fig. – 13, tabl. – 1, chair. – 16 , annexes. – 4, references – 50.

Keywords: cyber-physical systems; self-diagnostics; test automation; web applications, information and analytical subsystem.

The qualification work to the study of the development of a self-diagnostic subsystem in cyber-physical systems to improve their reliability.

The first chapter describes the features of CPS, considers testing problems, highlights automation methods, and analyses modern approaches to diagnostics.

The second chapter investigates system requirements, presents the subsystem architecture, and justifies the choice of technologies and modelling of its components.

The third chapter describes the implementation of the system, analyses its main modules, and tests its performance and functionality.

The fourth chapter examines ensuring safe operation of equipment.

Object of research: cyber-physical systems requiring automated self-diagnostics.

Subject of the study: an information and analytical subsystem for detecting and analysing failures in cyber-physical systems, built using automated testing methods and scenario language interpretation.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

API (англ. Application Programming Interface) – інтерфейс прикладного програмування.

DSL (англ. Domain-Specific Language) – предметно-орієнтована мова програмування.

JSON (англ. JavaScript Object Notation) – текстовий формат обміну даними, зручний для зберігання та передачі структурованої інформації.

MVC (англ. Model–View–Controller) – шаблон архітектури програмного забезпечення «модель–подання–контролер».

REST (англ. Representational State Transfer) – архітектурний стиль взаємодії систем через інтернет.

CRUD (англ. Create, Read, Update, Delete) – базові операції з базами даних: створення, читання, оновлення, видалення.

ANTLP (англ. ANother Tool for Language Recognition) – генератор парсерів для побудови синтаксичних аналізаторів.

KDT (англ. Keyword-Driven Testing) – методологія автоматизованого тестування, яка базується на використанні ключових слів для опису дій тестів

SUT (англ. System Under Test) – система, що тестується.

UI (англ. User Interface) – інтерфейс користувача.

UX (англ. User Experience) – досвід користувача.

FM (англ. Frequency Modulation) – частотна модуляція.

SNR (англ. Signal-to-Noise Ratio) – співвідношення сигнал/шум.

КФС – Кіберфізична система.

ПЗ – Програмне забезпечення.

ЗМІСТ

ВСТУП

Актуальність теми. Сучасні кіберфізичні системи (КФС) є основою багатьох промислових процесів, де від надійності їх функціонування залежить ефективність виробництва. Оскільки збої в роботі КФС можуть призводити до значних фінансових втрат, актуальним є створення інструментів для своєчасної самодіагностики. Розробка інформаційно-аналітичної підсистеми, здатної виявляти помилки у реальному часі, є важливим кроком до підвищення стабільності та безпеки таких систем.

Мета і задачі дослідження. Метою даної кваліфікаційної роботи освітнього рівня «Бакалавр» є підвищення якості послуг автоматизованого тестування шляхом розробки інформаційно-аналітичної підсистеми для самодіагностики в кіберфізичних системах. Для досягнення поставленої мети потрібно виконати ряд завдань, зокрема:

– проаналізувати сучасний стан і проблематику тестування КФС;

– дослідити архітектурні підходи до реалізації самодіагностики;

– спроєктувати архітектуру інформаційно-аналітичної підсистеми;

– реалізувати прототип системи клієнт-серверної моделі;

– забезпечити інтеграцію мови DSL та методології KDT для створення гнучких тестових сценаріїв;

– провести валідацію функціональності на основі тестових сценаріїв.

Практичне значення одержаних результатів.

Полягає в розробці працездатної інформаційно-аналітичної підсистеми для самодіагностики в кіберфізичних системах, яка дозволяє своєчасно виявляти збої у функціонуванні обладнання. Запропоноване рішення може бути інтегроване в реальні виробничі процеси для зменшення простоїв, підвищення надійності КФС і зниження витрат на діагностику та обслуговування. Розроблена система підтримує гнучке створення тестових сценаріїв, що дає змогу ефективно адаптувати її до різних типів обладнання та середовищ.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАВДАННЯ

1.1 Визначення умов

Виробництво багатьох промислових організацій підтримується кіберфізичними системами (КФС), які є інтеграцією обчислювальних, мережевих і фізичних процесів, і тому повинні досягати максимальної продуктивності у виробництві, оскільки використання КФС є необхідною умовою успіху виробництва в організації.

КФС інтегрує динаміку фізичних процесів з програмним забезпеченням і мережею, забезпечуючи абстракції, методи моделювання, проектування та аналізу для інтегрованого цілого. Ці системи повинні залишатися надійними і гарантувати свою функціональність [1]. Однак, щоб гарантувати, що ці системи функціонують правильно, необхідна їх регулярна діагностика. Для розробки КФС–тестів потрібні висококваліфіковані інженери, оскільки їх обчислювальна частина запрограмована на мовах низького рівня [2,3]. Традиційні тестові системи пристосовані до кожного конкретного випадку, що вимагає дуже дорогих і трудомістких зусиль для розробки, обслуговування або реконфігурації. Актуальним завданням є розробка інноваційних та реконфігурованих архітектур для тестових систем з використанням нових технологій та парадигм, які можуть дати відповідь на ці вимоги [4]. Завдання полягає в тому, щоб автоматизувати максимальну кількість завдань у цьому процесі та отримати максимальну віддачу від тестової системи, будь то КФС або просто програмне забезпечення.

CONTROLAR – це компанія, що займається розробкою апаратного та програмного забезпечення для промисловості, з величезним досвідом роботи в галузі автомобільних електричних компонентів і відмінним ноу–хау в розробці систем промислової автоматизації та функціональних і якісних випробувань електронних пристроїв. Одним з напрямків діяльності компанії є системне тестування, яке базується на розробці та інтеграції наявних тестових систем, моніторингу даних для перевірки електричних характеристик і тестів якості, а

також на використанні плат збору даних, генераторів і спеціального обладнання для збору та аналізу сигналів і даних.

З цією метою було створено партнерство з об'єднання навичок і знань Консорціуму між CONTROLAR, Університетом Міньо [5,6], Центром комп'ютерної графіки [7] та дослідницьким проектом під назвою "Інтелектуальні машини тестових систем" (TSIM). Цей проект спрямований на забезпечення створення цінності продукції CONTROLAR (як поточного постачальника Bosch) та її адаптацію до реалій Bosch Industry 4.0. Крім того, метою є розробка інструментів, які зроблять автоматичне випробувальне обладнання CONTROLAR більш гнучким, ефективним та інтелектуальним.

Одним із продуктів, які компанія CONTROLAR сьогодні постачає Bosch, є інтелектуальний функціональний

Випробувальна машина [8]. Ця машина є КФС, розробленою для проведення різних функціональних випробувань електронних пристроїв і компонентів. Bosch використовує цю машину в кінці виробничої лінії, щоб забезпечити правильну функціональність автомобільних радіоприймачів.

1.2 Визначення завдань

Як зазначалося вище, одним із продуктів, які CONTROLAR постачає компанії Bosch, є інтелектуальна машина функціонального тестування [8], КФС, розроблена для виконання різних рівнів функціональних випробувань електронних пристроїв і компонентів. Bosch використовує цю машину в кінці виробничої лінії, щоб забезпечити правильну функціональність автомобільних радіоприймачів. Таким чином, автомобільні радіоприймачі піддаються різним випробуванням під час використання цієї машини.

Проблема з'являється, коли машина виявляє помилки в декількох магнітолах, що вказує на те, що виробнича лінія вийшла з ладу на одному з її сегментів, що

може призвести до помилок у всіх магнітолах цієї лінії. Таким чином, сценарій, коли всі автомагнітоли поставляються з помилками, означає, що на виробництві стався якийсь збій. Це неприйнятно для компанії, оскільки призведе до непередбачуваних затримок. Більше того, виникає необхідність відремонтувати всі автомагнітоли, що призводить до збільшення витрат. Коли таке трапляється, організація схильна приписувати проблему машині, а не своїй продукції, що призводить до спроб виявити проблему в машині.

Інша проблема полягає в тому, що машина не має модуля, який би визначав, чи працює вона правильно, чи ні. Єдиний спосіб дізнатися це – змінити фізичні зв'язки та внутрішні властивості конструкції машини, щоб спробувати зрозуміти причину несправності. Здебільшого це відбувається тому, що люди намагаються шукати помилки в машині і, як правило, завжди в кінцевому підсумку завдають їй шкоди, не усвідомлюючи цього, тому що вони не знають всіх деталей її конструкції і починають змінювати багато чого, поки не зроблять машину повністю непридатною для використання, так і не знайшовши проблему. Ймовірним результатом таких дій є необхідність ремонту машини або навіть її заміни, що призводить до фінансових втрат.

Ця робота зумовлена необхідністю знайти рішення для розв'язання поставленої проблеми. Це також є інноваційним рішенням, яке є внеском у світ досліджень в області КФС і тестових систем самодіагностики. Відповідь полягає в тому, щоб інтегрувати в машину систему самодіагностики, яка може перевірити функціональність пристрою. Коли з'являються такі збої в роботі автомагнітол, Bosch може бути впевнений, чи проблема дійсно в машині, чи в лінії виробництва автомагнітол. Оскільки пристрій є КФС, він дозволяє інтегрувати програмну систему, яка може контролювати та керувати всіма його діями. Тому необхідно розробити систему тестування для замовлення тестів та їх виконання, виявлення внутрішніх збоїв машини. Однак, перш ніж ми зможемо створити план, важливо знайти правильну архітектуру для системи тестування. Ця архітектура повинна максимально відповідати проблемі, з якою ми стикаємося, і бути максимально загальною щодо того, що таке система тестування.

Розробка системи самодіагностики для інтеграції в КФС, здатної виконувати самодіагностику КФС в режимі реального часу, що забезпечує її цілісність, є основною метою статті. В якості конкретних завдань, що стосуються аналізу та специфікацій системи, ми розробляємо та пропонуємо нову архітектуру для управління та конфігурації тестів, а також нову архітектуру для тестової системи самодіагностики, пропонуємо архітектуру для інтеграції тестової системи самодіагностики в КФС, розробляємо тестову систему самодіагностики та валідизуємо тестову систему самодіагностики.

Що стосується інтеграції системи тестів для самодіагностики, то методологія дослідження повинна включати вивчення проблеми, огляд сучасного стану, роздуми про технології, які будуть використовуватися при розробці, розробку архітектури системи і, нарешті, розробку, валідацію та інтеграцію системи.

1.3 Аналіз стану концепції КФС

КФС – це інтеграція обчислювальних, мережевих та фізичних процесів, що поєднує динаміку фізичних процесів з програмним забезпеченням та Інтернетом, надає абстракції та методи моделювання, проектування та аналізу для інтегрованого цілого. Однак, щоб реалізувати їхній повний потенціал, основні концепції обчислень потребують переосмислення та вдосконалення [1].

Наразі КФС потребує рішень, які підтримують її на рівні пристроїв, систем, інфраструктури та додатків. Це виклик, який включає в себе інженерний підхід та поєднання комунікаційних, інформаційних технологій та технологій автоматизації [9]. Для ефективної оркестровки програмних і фізичних процесів необхідні семантичні моделі, які відображають відповідні властивості обох процесів [1]. Поточний виклик полягає в розробці інноваційних, гнучких і реконфігурованих архітектур для систем управління, використовуючи нові технології та парадигми, які можуть забезпечити відповідь на вимоги [4]. При цьому все ще необхідно

підтримувати баланс між потребами і поняттям легких і безпечних рішень [10]. Розробка програмного забезпечення для цієї галузі є складним завданням [11].

Інфраструктура зв'язку має важливе значення для тестування. Більшість зосереджується на комунікаційно–орієнтованих дослідженнях, конфіденційності та безпеці інфраструктури [12,13]. Тести повинні бути розроблені таким чином, щоб забезпечити віддалений інтерфейс [14]. Методи верифікації та тестування композиції також повинні бути адаптовані до КФС. Створення автоматизованого або напівавтоматичного процесу для оцінки результатів системних тестів є складним завданням при тестуванні КФС [15]. Існує ще багато обмежень для більш широкого промислового застосування КФС–тестування [16]. Завдання полягає в тому, щоб автоматизувати максимальну кількість завдань і отримати максимальну віддачу від КФС.

1.4 Автоматизація роботи з текстом

Автоматизація тестування має вирішальне значення для якості кінцевого продукту. Таким чином, розробники програмного забезпечення зобов'язані мати набір стандартів якості на всіх етапах розробки проекту, оскільки організації, приймаючи програмне забезпечення, покладаються на критерії якості, оскільки одним із основ у забезпеченні якості програмного забезпечення є тестування [17].

На ринку розробки актуальною проблемою є створення якісного програмного забезпечення, оскільки метою тестувальника програмного забезпечення є визначення реалізації цього програмного забезпечення, яке відповідає всім специфікаціям та очікуванням, а також автоматизація тестування, усвідомлюючи, коли і де має бути проведене тестування, що дає команді більше часу для планування тестувань. Крім того, автоматизація призводить до механізації всього процесу моніторингу та управління потребами в тестуванні та оцінці, пов'язаними з розробкою програмного забезпечення [18].

1.5 Виявлення збоїв

Виявлення збоїв – це дослідження збоїв, присутніх у компонентах машин. Нещодавнє дослідження показало, що більшість досліджень з виявлення збоїв зосереджені на початкових збоях, щоб можна було виконати наступний етап процесу FDD [19].

Метою виявлення збоїв є "виявлення виникнення збоїв у функціональних одиницях процесу, які призводять до небажаної або неприпустимої поведінки всієї системи". Більшість сучасних методів автоматизованого виявлення збоїв базуються на моделях, будь то моделі перших принципів, моделі, засновані на правилах (правила "якщо–то"), або моделі машинного навчання, засновані на декількох виміряних змінних.

Виявлення збоїв також може здійснюватися вручну. Фактично, звичайне обслуговування на основі скарг можна розглядати як форму ручного обслуговування на основі виявлення збоїв, оскільки скарги є похідними від спостережуваного стану Х, такого як температура, або параметра Ф, такого як працездатність вікон, що надходять від мешканців. Крім виявлення аномальних операцій, виявлення збоїв може надати цінну інформацію для подальшої діагностики збоїв [20].

1.6 Архітектурні особливості KDT, DSL, ANTLR та REST

KDT – це тип методології автоматизованого тестування, відомий як табличне тестування або тестування на основі дій, що використовує формат таблиці

(електронної таблиці) для визначення ключових слів або слів дій, які представляють зміст тестів. KDT представляє себе як цінний метод тестування для підтримки вимог до тестування промислового програмного забезпечення для управління [16]. Однак нещодавні результати досліджень показали, що структура тестів KDT є складною, з багатьма рівнями абстракції, і що ця структура сприяє повторному використанню, що потенційно може зменшити кількість змін, необхідних під час еволюції [21]. Крім того, ключові слова змінюються з відносно низькою швидкістю, що вказує на те, що після створення ключового слова вносяться лише локальні та уточнені зміни. Однак ті ж результати також показали, що методи KDT потребують інструментів для підтримки вибору ключових слів, рефакторингу та виправлення тестів [22].

- Поточні інструменти

Для написання роботи було використано деякі інструменти автоматизації тестування, що реалізують фреймворк тестування за ключовими словами, такі як Selenium [23], QuickTest Professional [24,25], TestComplete [26], SilkTest [27,28], Ranorex [29] та Robot Framework [30].

DSL – це предметно-орієнтована мова програмування, призначена для використання в контексті певної предметної області і є досить потужною для представлення та вирішення проблем і рішень у цій сфері [31]. Використовується для генерації вихідного коду за ключовим словом, плюс генерація коду з DSL не є обов'язковою. Деякі дослідники використовували DSL в КФС і залишили свої свідчення про те, як мова специфікацій приховує деталі реалізації. Специфікації автоматично збагачуються продуктивністю за допомогою багаторазових правил відображення. Ці правила реалізуються розробниками і визначають порядок виконання модулів та спосіб реалізації вхідних/вихідних змінних [32]. Це дозволяє повторно використовувати програмні компоненти та підвищує продуктивність і якість програмного забезпечення [33].

ANTLR – є синтаксичним аналізатором [34], оскільки він бере текст і перетворює його в організовану структуру, дерево, яке називається абстрактним синтаксичним деревом (АСТ) [35], яке схоже на історію, що описує зміст коду

(логічне представлення), створеного шляхом з'єднання різних частин [36], як це наведено на рисунку 1.1.

Рисунок 1.1 – Схема мовного процесора

Процес синтаксичного аналізу, показаний на рисунку 1.1, проходить три етапи: лексичний аналіз, синтаксичний аналіз і трансформації.

ANTLR – це генератор синтаксичних аналізаторів, який динамічно аналізує вхідні дані під час виконання, використовуючи синтаксичний аналізатор зверху вниз зліва направо, будуючи крайню ліву похідну інформації та розглядаючи будь–яку кількість токенів заздалегідь при виборі між альтернативними правилами [37]. Крім того, патерн відвідувача дозволяє нам вирішувати, як проходити по дереву і які вузли ми будемо відвідувати, дозволяючи нам визначати, скільки разів ми побачимо вузол [38].

REST – це архітектурний стиль між веб–додатками, що полегшує комунікацію між системами [39]. Це протокол, розроблений спеціально для створення веб–сервісів, який має на меті бути незалежним від платформи та мови, використовуючи протокол передачі гіпертексту (HTTP) для зв'язку між серверами. Цей протокол базується на шести фундаментальних принципах, а саме: клієнт–серверна архітектура, бездержавність, кеш, уніфікований інтерфейс, багаторівнева система та код на вимогу [40,41].

1.7 Висновок до першого розділу

У першому розділі проведено аналіз предметної області кіберфізичних систем (КФС) з акцентом на проблематиці самодіагностики та тестування їх функціональності. Визначено ключові виклики, пов’язані з виявленням збоїв у роботі тестових машин, а також з недосконалістю існуючих підходів до діагностики, що можуть спричинити додаткові витрати та зниження ефективності виробництва. Розглянуто актуальні технології автоматизованого тестування, зокрема методологію KDT, предметно-орієнтовані мови (DSL), синтаксичні аналізатори (ANTLR) і REST-архітектуру.

Обґрунтовано доцільність розробки інформаційно-аналітичної підсистеми для самодіагностики, яка б забезпечувала ефективне управління тестами, конфігурацію сценаріїв та визначення причин збоїв. Сформульовано мету дослідження та окреслено завдання, реалізація яких дозволить створити гнучку та масштабовану систему для автоматизованого виявлення несправностей у КФС.

РОЗДІЛ 2. ПРОЕКТНА ЧАСТИНА. АНАЛІЗ ТА СПЕЦИФІКАЦІЯ

2.1 Структура системи

Виходячи з мети роботи, яка полягає у створенні системи, що дозволяє проводити самодіагностику КФС шляхом виконання тестів, ця система повинна також мати можливість керувати виконаними тестами та отриманими результатами. Виходячи з цього припущення, виокремлено набір системних вимог, організованих у загальні потреби, специфічні вимоги, а також категорії користувачів та дозволи.

Після детального розгляду вимог до системи, ми представляємо огляд структури системи. В результаті аналізу вимог було обрано рішення для проектування цієї системи – клієнт–серверна архітектура, що базується на моделі REST. Клієнт–серверна модель має на меті розділити завдання для зменшення навантаження на систему.

Сервер буде пропонувати конкретному користувачеві ряд послуг, інтерфейс прикладного програмування (API), і буде виконувати завдання, запитувані користувачем, і повертати дані. З іншого боку, клієнт відповідає за запит певної послуги від сервера за допомогою повідомлень.

Система складатиметься з клієнта та сервера, де клієнт спілкуватиметься з сервером за допомогою HTTP–запитів, а сервер відповідатиме за допомогою HTTP–відповідей. Сервер відповідатиме за підключення до бази даних, виконання транзакцій або запитів і повернення результатів.

2.2 Клієнтська та серверна частина

Після аналізу вимог та структури системи, в цьому розділі описано обрані технології для розробки та реалізації системи. Систему можна розділити на два основні компоненти: клієнтську частину (фронтенд) та серверну частину (бекенд).

- Клієнтська частина

Цей компонент системи включатиме сервер API та базу даних, що використовується для забезпечення узгодженості даних системи. Після того, як було вирішено, що система повинна бути реалізована у вигляді веб–додатку, було доцільно використати платформу, яка працює в Інтернеті та мову JavaScript Pluralsight2021JavaScript.

Для цього було обрано Node.js Foundation2021Node.js. Використання Node.js дозволить додатку бути швидким і масштабованим, а для цього платформа покладається на неблокуючий ввід/вивід, асинхронне програмування, кероване подіями, і один потік.

Для забезпечення узгодженості даних було обрано базу даних MongoDB, яка є базою даних документів; вона зберігає дані в документах типу JavaScript Object Notation (JSON). Оскільки дані завжди будуть передаватися у вигляді JSON–документів, використання MongoDB дозволить підтримувати цю узгодженість у більш природний, виразний і потужний спосіб, ніж будь–яка інша модель [42].

Вона управляє зв'язками між даними, забезпечує валідацію схеми та здійснює переклад між об'єктами в коді та представленням цих об'єктів у MongoDB. Наприклад, на рисунку 2.1 наведено це відображення об'єктів між Node.js та MongoDB за допомогою Mongoose.

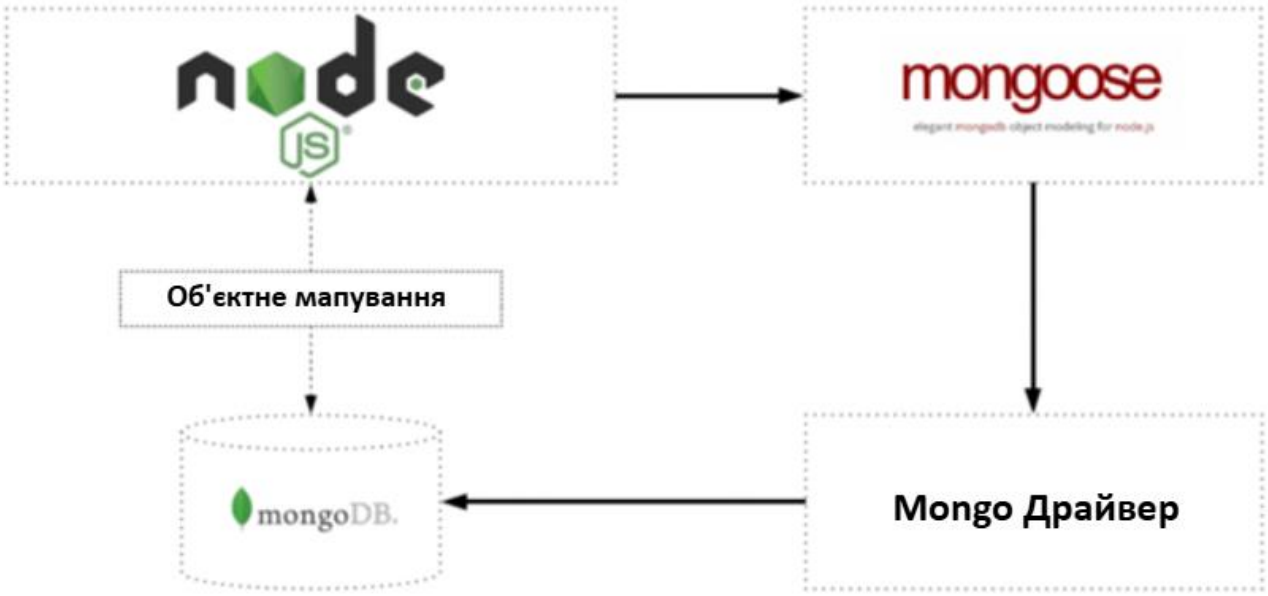

Рисунок 2.1 – Об'єктне мапування між Node.js та MongoDB, кероване через Mongoose

- Серверна частина

Для цієї системи було обрано React.js [43] завдяки його здатності взаємодіяти з додатками, розробленими за допомогою технологій, обраних для Backend [44]. На React.js легко створити інтерактивний інтерфейс користувача, оскільки він проектує прості візуалізації для кожного стану додатку та ефективно оновлює і рендерить потрібні компоненти при зміні їхніх даних [45]. Більше того, React.js дозволяє створювати інкапсульовані компоненти, які керують своїм станом. Завдяки композиції цих компонентів можна створювати більш складні інтерфейси з меншою складністю коду.

Оскільки логіка компонента написана на JavaScript, а не на моделях, ми можемо передавати дані через додаток і підтримувати стан за межами DOM [46]. Крім того, віртуальний DOM React дозволяє реалізувати деякі інтелектуальні альтернативні рішення, які гарантують швидкий рендеринг компонентів, що є необхідним, оскільки дані повинні бути представлені негайно та ефективно. У таких ситуаціях React знаходить ідеальний спосіб оновити інтерфейс користувача, і все, що потрібно зробити – це забезпечити потік даних через API [47]. Віртуальний DOM React діє як проміжний етап, коли на веб–сторінці відбуваються зміни. Це

дозволяє рендерингу бути швидшим та ефективнішим, роблячи сторінки дуже динамічними. Різницю між Real DOM та React Virtual DOM можна побачити на рисунку 2.2.

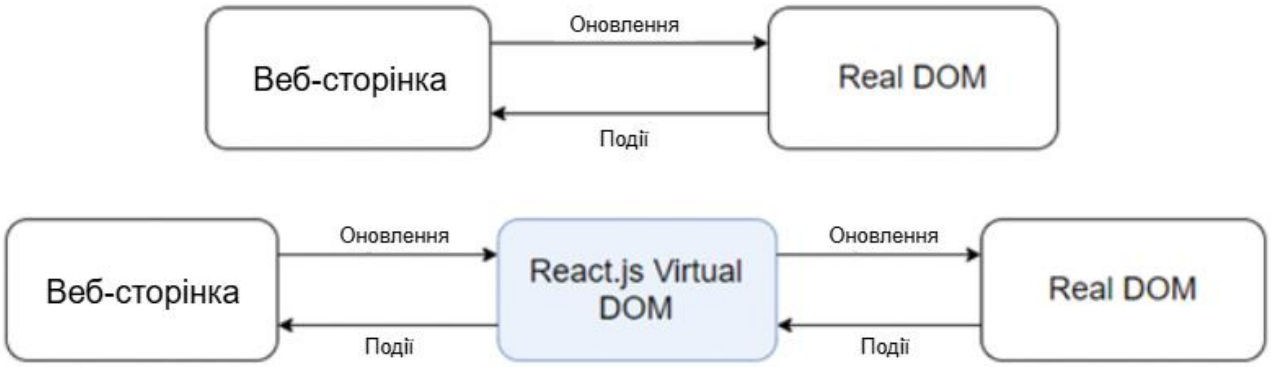

Рисунок 2.2 – Реальний DOM та React Virtual DOM

2.3 Загальна архітектура КФС

2.3.1 Архітектура управління тестуванням та конфігурацією

Архітектура, яку ми пропонуємо у цьому розділі, спрямована на автоматизацію та полегшення нових тестів. У цій архітектурі ми використовуємо методологію KDT у поєднанні з DSL. Для повного розуміння архітектури та взаємозв'язку між її компонентами, ми пояснимо, як застосовуються KDT та DSL, а потім, як вони інтегровані в одну архітектуру.

- Методологія тестування на основі ключових слів

KDT буде використовуватися для абстрагування низькорівневих скриптів коду, пов'язуючи кожну руку з ключовим словом, яке представляє її якомога більш описово і явно, що полегшує роботу користувача. Пов'яжемо кожне ключове слово з метаданими, пов'язаними з відповідним тестом, що зберігається в базі даних. На рисунку 2.3 показано підхід, який ми застосовуємо у використанні KDT. Назви труднощів на рисунку є вигаданими, щоб показати, що слова, які відповідають ключовим словам, повинні бути якомога більш описовими.

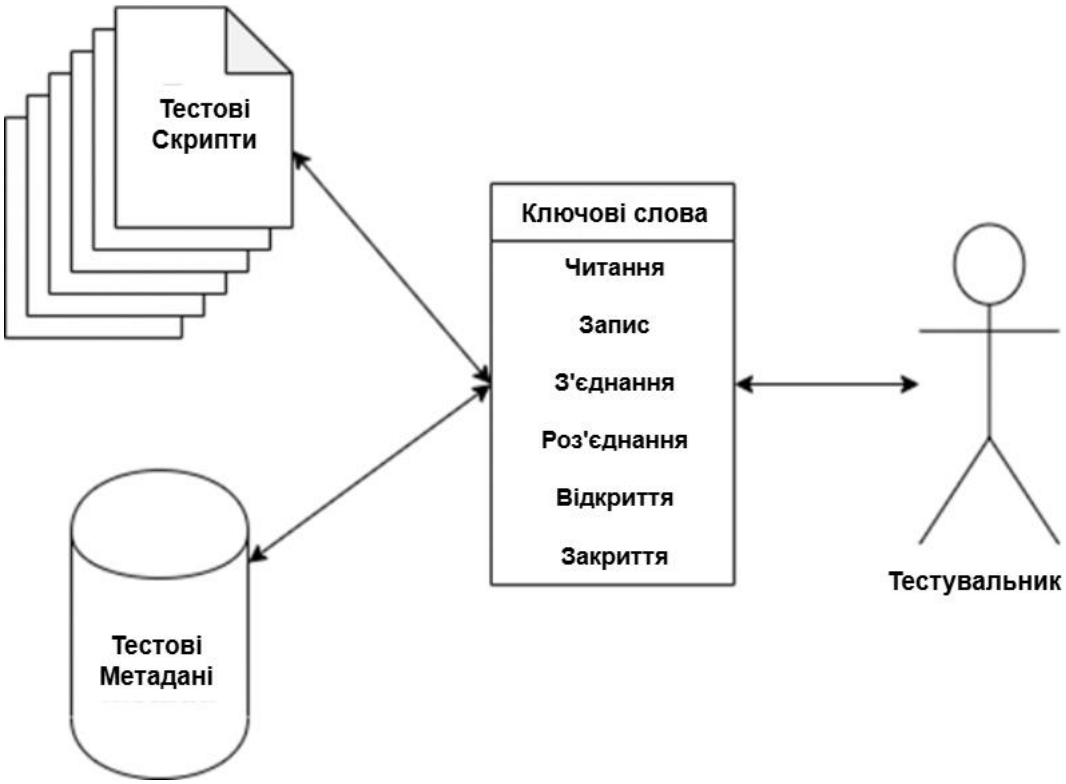

Рисунок 2.3 – Підхід KDT

На рисунку 2.3 ми бачимо стек скриптів, які представляють примітивний тест в системі, який фокусується лише на перевірці функцій, доки вони можуть бути добре ідентифіковані лише за допомогою слова, яке може слугувати ключовим словом. Ми також бачимо представлення бази даних, яка буде зберігати всю інформацію та метадані про тести, що існують в системі. Підключившись до бази даних і стеку скриптів, ми бачимо таблицю з ключовими словами, що представляє всю інформацію, пов'язану з випробуванням, яка є найбільш важливим елементом на малюнку, оскільки саме тут ми можемо дізнатися всі дані зі стеку скриптів і бази даних. Це робиться за допомогою лише одного слова, що дозволяє тестувальникам з невеликими знаннями програмування інтерпретувати, що робить або означає кожен тест. Нарешті, ми маємо зв'язок між тестувальником і таблицею ключових слів, який демонструє, що він матиме доступ лише до ключових слів, не знаючи жодних деталей реалізації.

- Предметно-орієнтована мова програмування

Використання KDT не дає переваг, оскільки нам потрібно щось для проектування потоку виконання тесту. Для цього у нас є DSL, що забезпечує дружню мову для тестувальників без необхідності знання програмування. Запропонована мова проста, але дозволяє створювати нові сценарії з новими потоками виконання і логічними правилами. Це досягається лише використанням ключових слів, визначених KDT, та деяких символів терміналів, визначених у DSL. У таблиці 1 показано термінологічні символи DSL і те, що вони означають.

Таблиця 1.1 – Символи DSL.

Символ	Опис
keyword	Ловить ключові слова в скрипті
–>	Перехоплює символ "next", що означає, що після цього символу надходить наступний блок для виконання
(	Перехоплює відкриваючу дужку
)	Перехоплює закриваючу дужку
?	Перехоплює умовні вирази зі скрипта
:	Перехоплює наступний блок коду, який виконується, коли умова є хибною
&	Перехоплює логічний оператор, що означає перетин
|	Перехоплює логічний оператор, що означає об'єднання
;	Перехоплює кінець сценарію

- Архітектура

Остаточна абстракція всіх цих процесів необхідна для досягнення повного потенціалу інтеграції між KDT та DSL. На рисунку 2.4 представлено архітектуру, яка гарантує абстрагування всього складного процесу створення нових тестів для системи, таким чином надаючи можливість користувачам, менш наділеним знаннями програмування, створювати нові тести.

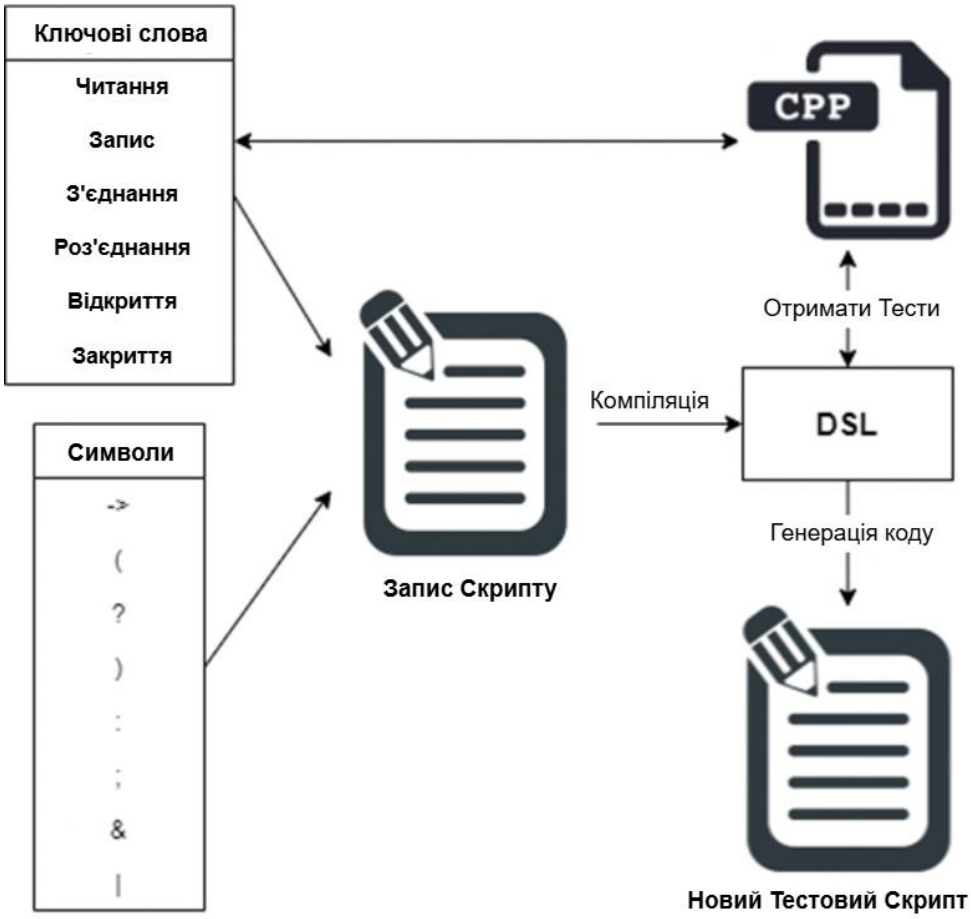

Рисунок 2.4 – Пропонована архітектура для KDT з DSL

За допомогою ілюстрації на Рисунку 2.4 можна перевірити дві таблиці, що представляють ключові слова і символи, які використовуються для формування нових тестових сценаріїв, представляючи в якості інгредієнтів ключові слова, які відповідають визначеним тестам і доступні в системі для використання при створенні нових тестів.

За допомогою того ж рисунку 2.4 можна перевірити зв'язок між існуючими труднощами з таблицею ключових слів, оскільки елементи таблиці символів представляють собою термінальні символи, визначені в DSL. Це дозволяє нам надати організацію і логіку новим тестам, і, таким чином, за наявними елементами в двох таблицях, стає можливим написати новий тестовий сценарій, який демонструє зв'язок між компонентом Write Script і таблицями.

Щойно новий тестовий скрипт буде написано, DSL проаналізує його за допомогою лексеми та синтаксичного аналізатора і перевірить, чи правильно він

написаний з синтаксичної та лексичної точок зору. Цей крок представлений у з'єднанні компіляції. Якщо скрипт відповідає визначеним правилам, DSL скомпілює цей скрипт і згенерує код для нового тесту. Однак для цього йому потрібен доступ до принципу роботи тестів за допомогою ключових слів, що представлено з'єднанням Get Tests. Наприкінці цього процесу DSL згенерує код для нового тесту. Це описано у посиланні «Згенерувати код». З цього моменту новий тест доступний для виконання в системі.

2.3.2 Архітектура тестової системи самодіагностики

- Архітектура з’єднання клієнтської та серверної частин

Тут ми представили відповідну архітектуру з'єднання фронтенду та бекенду, яка відповідає системі самодіагностики, що буде інтегрована в КФС, як показано на рисунку 2.5.

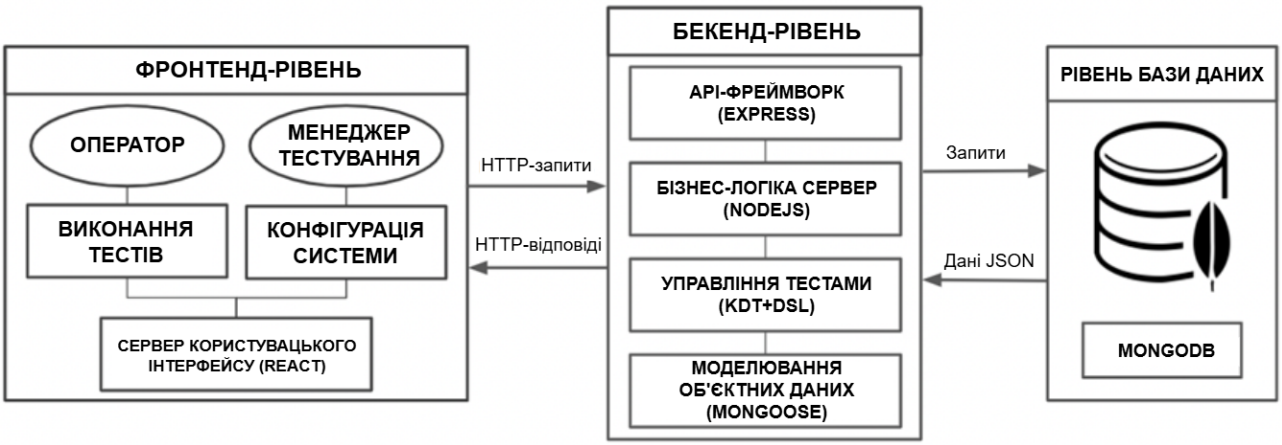

Рисунок 2.5 – Архітектура тестової системи самодіагностики

У цій архітектурі представлені три основні рівні системи. Фронтенд–рівень, який взаємодіє з бекенд–рівнем, відповідає за логіку системи за допомогою HTTP–запитів і відповідей. Інтерфейсний рівень взаємодіє з рівнем бази даних, який відповідає за узгодженість даних системи, за допомогою звернень у вигляді запитів

і відповідей у вигляді JSON–документів, які обробляються і перевіряються за допомогою об'єктного моделювання даних. Найважливішим моментом цієї архітектури, який відрізняє її від інших, є включення та інтеграція архітектури управління та конфігурації тестів, яка разом з іншими компонентами системи дозволить системі забезпечити свою цілісність і функціональність за допомогою самодіагностики в режимі реального часу і конфігурувати нові тести для системи з меншою складністю.

- Загальна архітектура кіберфізичної системи

Представлено та пояснено остаточну архітектуру КФС, в якій ми інтегруємо всі її компоненти з системою тестів самодіагностики. Ця архітектура має на меті забезпечити можливість самодіагностики та, таким чином, ідентифікувати збої в роботі у випадку будь–якої внутрішньої помилки. Остаточна архітектура системи показана на Рисунку 2.6 і пояснюється нижче.

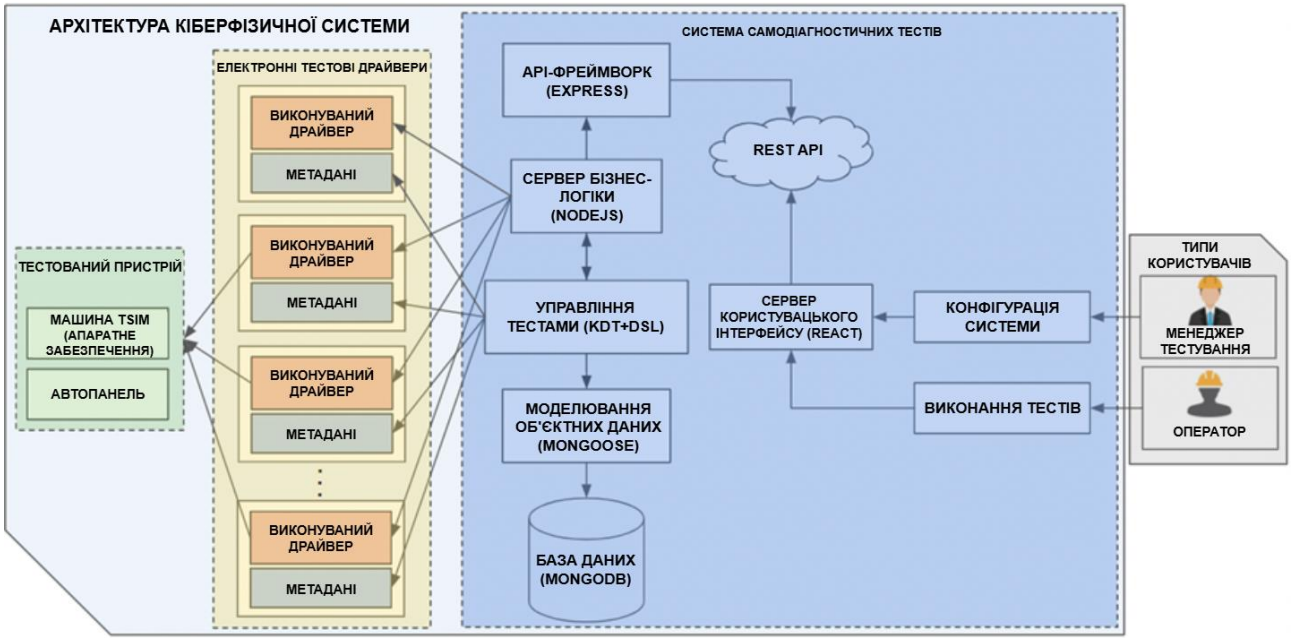

Рисунок 2.6 – Архітектура тестової системи самодіагностики, інтегрованої з КФС

У цій архітектурі можна швидко виділити чотири групи компонентів. Три з них становитимуть невід'ємну частину КФС: пристрої, що тестуються, електронні тестові драйвери та система самодіагностики. Остання група, типи користувачів, буде важливим втручанням, але вона не є невід'ємною частиною КФС.

Група пристроїв, що тестуються, містить пристрої, які можуть бути піддані тестуванню: автомобільні радіоприймачі та сама машина. Елементи автомобільна радіостанція та машина TSIM представляють два типи пристроїв.

Група електронних тестувальників відповідає за примітивні тести системи, які можуть бути будь–якими, якщо вони дотримуються одного формату інтеграції. Тому кожен елемент цієї групи повинен дотримуватися наступного формату:

– Виконуваний драйвер – надає виконуваний файл драйвера для запуску, який містить кілька примітивних тестів, які можна запустити і протестувати тестовані пристрої;

– Метадані – надає файл метаданих, що містить всю інформацію про тести водія.

У групі Тестова система самодіагностики представлена розроблена в цій роботі система, яка дозволить користувачам керувати та виконувати системні тести. Елемент TEST MANAGEMENT відповідає за завантаження всіх метаданих примітивних тестів, наявних у файлах метаданих електронного контролера тестів та збережених у базі даних системи для виконання.

Елементом зв'язку з базою даних є об'єктна модель бази даних, яка буде зв'язувати і обробляти запити і транзакції з базою даних, яка є елементом бази даних. Управління тестуванням здійснюється за допомогою методології KDT, описаної раніше, а конфігурація нових тестових запусків здійснюється за допомогою розробленого DSL, також описаного раніше.

Тести будуть виконуватися сервером бізнес–логіки, який отримуватиме замовлення на виконання від кінцевого користувача. Цей сервер знатиме, які тести доступні в кожному драйвері, оскільки елемент TEST MANAGEMENT вже зібрав метадані всіх драйверів і на той момент зробив всі тести, що містяться в них, доступними для виконання.

Сервер бізнес–логіки контролює всі дані і напрямок роботи системи, а також визначає маршрути і типи запитів, які може робити клієнтська частина. Він представляє послуги, які будуть доступні, і це називається API. Фреймворк API

відповідає за створення та забезпечення доступу до API для будь–якого клієнта з відповідними правами, які також визначаються сервером бізнес–логіки.

Сервер користувацького інтерфейсу представляє клієнтську частину, відповідальну за створення веб–інтерфейсу для кінцевих користувачів. Він робить HTTP–запити із зазначенням сервісів, до яких він хоче отримати доступ, щоб отримати дані, необхідні для своїх сторінок.

Доступні два типи інтерфейсів: інтерфейс виконання, представлений елементом TESTS EXECUTION, та інтерфейс управління тестами і конфігурацією, представлений елементом SYSTEM CONFIGURATION, з відповідним користувачем, що підводить нас до останньої групи, визначеної в архітектурі, Типи користувачів.

Ця група представлена елементом user types і описує різні типи користувачів кінцевої системи. Першим і найпростішим типом користувача є оператор. Цей промисловий оператор працює і керує КФС і запускає тільки тести або тестові пакети системи. Другий тип користувача, вже більш складний, – це менеджер тестування, який відповідає за управління всією системою, використовуючи для цього відповідний інтерфейс.

2.4 Висновок до другого розділу

У другому розділі здійснено проєктування інформаційно-аналітичної підсистеми самодіагностики для кіберфізичних систем. На основі аналізу вимог запропоновано архітектуру системи, що ґрунтується на клієнт-серверній моделі з використанням REST API. Обґрунтовано вибір технологій для фронтенд- і бекенд-частин, зокрема застосування React.js, Node.js та бази даних MongoDB.

Представлено дві ключові архітектурні моделі: архітектуру управління тестуванням із використанням методології KDT та DSL для побудови гнучких

тестових сценаріїв, а також архітектуру самодіагностичної системи з урахуванням інтеграції до КФС.

Описані основні функціональні компоненти системи, їх взаємозв’язки та призначення, а також визначено ролі користувачів і принципи управління тестами. Запропоноване рішення є масштабованим і придатним до використання в умовах сучасного виробництва.

РОЗДІЛ 3. ПРАКТИЧНА ЧАСТИНА. РЕАЛІЗАЦІЯ ТЕСТОВОЇ СИСТЕМИ САМОДІАГНОСТИКИ

3.1 База даних

Для бази даних було використано MongoDB, яка є базою даних документів. Тобто вона зберігає дані у вигляді JSON–документів. Згідно з даними системи, було визначено п'ять колекцій даних для зберігання в базі: конфігурації, тести, звіти, розклади та пакети.

Колекція конфігурації містить атрибути, які можуть відрізнятися на різних машинах і необхідні для забезпечення коректної роботи системи. Нарешті, колекція тестів зберігає всі метадані, надані тими, хто створює і робить доступними примітивні тести. Вони імпортуються до бази даних системи та оновлюються.

Колекція звітів зберігає всі звіти про виконання примітивних тестів або тестових пакетів в системі. У колекції schedules зберігаються всі виконання примітивних тестів або наборів тестів, запланованих користувачем на певний час. Це не стосується колекції packages, яка зберігає всі метадані для нових наборів тестів, створених у системі з примітивних тестів.

Після визначення даних, які мають бути збережені в кожній колекції бази даних системи, наступний розділ пояснює, як система взаємодіє з базою даних за допомогою запитів для отримання даних, необхідних для її роботи.

3.2 Бекенд

Бекенд – це рівень системи, який відповідає за управління базою даних та надання даних інтерфейсу користувача. Таким чином, побудований за архітектурою Model–View–Controller (MVC), він є контролером системи і встановлює зв'язок між базою даних і користувацькими інтерфейсами, гарантуючи

цілісність даних, не дозволяючи іншим компонентам отримати до них доступ або змінити їх.

Технологія, яка використовувалася для розробки цього сервера – Node.js у поєднанні з Framework Express. Цей сервер організований таким чином, що існує поділ коду відповідно до його функції. Замість того, щоб весь код знаходився в одному файлі, він був розділений по різних файлах і директоріях відповідно до його призначення на сервері. Це дозволить повторне використання та модульність розробленого коду, що полегшить його обслуговування та розуміння в майбутньому.

Таким чином, структура сервера складається з моделей, відповідальних за колекції, що зберігаються в базі даних, контролерів, відповідальних за виконання всіх системних операцій, граматики, яка відповідає DSL, розробленій для системи, маршрутів, які пересилають запити від клієнта, і, нарешті, app.js, що активується через модуль express.

Кожен з цих елементів відіграє важливу роль у логіці роботи сервера і буде детально описаний нижче для кращого розуміння.

3.2.1 Моделі

Моделі представляють колекції, що зберігаються в базі даних. Потім кожна модель повинна представляти свою колекцію і перевіряти типи даних своїх атрибутів перед тим, як виконувати транзакції з базою даних.

У ці моделі імпортується модуль "mongoose" – об'єкт моделювання даних, який дозволяє підключатися до бази даних в асинхронному середовищі та надсилати і отримувати дані у форматі JSON, що полегшить використання даних у системі, як показано у лістингу 3.1.

```
const mongoose = require('mongoose');

const testsSchema = new mongoose.Schema(
    {
        id_test: { type: mongoose.Schema.Types.ObjectId, required:
true },
        module: { type: String, required: true },
        name: { type: String, required: true },
        result: { type: String, required: true },
        message: { type: String, required: false },
        runtime: { type: Number, required: true },
        resultValue: { type: String, required: false },
    },
    { versionKey: false }
);

const reportSchema = new mongoose.Schema(
    {
        id_user: { type: String, required: true },
        date: { type: String, required: true },
        results: [testsSchema],
    },
    { versionKey: false }
);

module.exports = mongoose.model("report", reportSchema);
```

Лістинг 3.1 – Модель звіту.

У рядку 1 лістингу 3.1 ми бачимо імпорт модуля "mongoose". Далі, між рядками 3 і 14, ми бачимо модель у вигляді об'єкта, що представляє структуру результату тесту з його атрибутами і типами даних. Цей об'єкт слугує допоміжною структурою, в даному випадку для введення в модель звіту. Далі, між рядками 16 і 23, ми бачимо модель звіту у вигляді об'єкта з його атрибутами і типами даних. Потім, у рядку 20, де вказано атрибут "результати", ми бачимо список речей, які є об'єктами з визначеною вище структурою, для наслідків кожного тесту. Нарешті, в рядку 25 ми бачимо експорт створеної моделі, що робить її доступною для використання в інших файлах. У цьому випадку вона буде використана контролером, який буде відповідати за операції збору звітів.

3.2.2 Граматика

Мова DSL, розроблена в цій статті, має на меті створення нових наборів тестів з примітивних тестів, доступних в системі, із застосуванням правил і логіки, які повинні виконуватися за найкоротший час. Мова була створена з символів терміналу для ідентифікації лексером. Синтаксичний аналізатор був створений для правил логіки та побудови речень зазначеної граматики. Термінальні символи наведено у таблиці 1. Структура лексера наведена нижче у лістингу 3.2:

```
lexer grammar TestLexer;

NEXT: '->';
AND: '&';
OR: '|';
IF: '?';
ELSE: ':';
RPAREN: ')';
LPAREN: '(';
END: ';';
KEYWORD: ([A-Za-z]+ ([/_ -] [A-Za-z]+)*);
WS: [\r\n\t]+ -> skip;
```

Лістинг 3.2 – Граматичний аналізатор.

Структура лексера досить проста, починаючи з його ідентифікації, а потім визначення всіх термінальних символів, які повинні бути розпізнані. Ці символи визначаються за допомогою регулярних виразів, завжди гарантуючи, що це визначення не містить несподіваних елементів і, отже, не є двозначним.

Символи, які ми бачимо в цій граматиці, дуже інтуїтивно зрозумілі і легкі для розуміння кінцевим користувачем, що є однією з цілей. Єдиний символ, який дає додаткові пояснення – це символ ключового слова. Цей символ повинен розпізнавати всі назви примітивних тестів, введених у скрипті. Крім того, його регулярний вираз включає окремі слова, що дає користувачеві певну свободу у виборі ключових слів. Після визначення всіх цих факторів настав час задати

правила побудови речень з цими символами. Це робиться за допомогою синтаксичного аналізатора, наведеного нижче у лістингу 3.3:

```
parser grammar TestParser;

options {
    tokenVocab=TestLexer;
}

test
    : statement END
    ;

statement
    : condition #Conditional
    | seq #Sequence
    ;

condition
    : expr IF statement ELSE statement #IfElse
    | expr IF statement #If
    ;

seq
    : KEYWORD (NEXT statement)*
    ;

expr
    : LPAREN KEYWORD (AND KEYWORD)* RPAREN #And
    | LPAREN KEYWORD (OR KEYWORD)* RPAREN #Or
    ;
```

Лістинг 3.3 – Граматичний аналізатор.

Синтаксичний аналізатор починається з ідентифікації, після чого за посиланням на лексер, який надає символи, визначається, які з них є термінальними. Після цих двох кроків визначаються речення граматики. В операторі елемента ми бачимо умову, що представляє умовний вираз, і послідовність, що представляє послідовність тестів. Найважливішою частиною синтаксичного аналізатора, яку потрібно зберегти, є елементи, що знаходяться в кінці рядків для кожної можливості, визначеної на початку слів за допомогою символу #. Це дозволяє відвідувачу знати можливі шляхи у дереві синтаксичного аналізу, які згенерує цей синтаксичний аналізатор.

Для того, щоб система могла використовувати цю граматику, необхідно знайти спосіб її використання в системі. Оскільки ANTLR пропонує трансформацію граматики для декількох мов програмування, ми перейдемо до трансформації граматики в JavaScript і включимо код безпосередньо в систему. Для цього необхідно виконати наступну команду:

```
$ antlr4 -Dlanguage = JavaScript Lexer.g4 Parser.g4 -no-listener -visitor
```

У цій команді ми вказуємо лексер і синтаксичний аналізатор, які потрібно трансформувати.

Після цього ми визначаємо генерацію відвідувача. Після виконання цієї команди буде згенеровано декілька файлів, у тому числі і відвідувач.

Саме тут буде вказано код, який буде розроблено для нових наборів тестів, як ми бачимо нижче, у лістингу 3.4.

```
TestParserVisitor.prototype.visitAnd = function(ctx) {
    this.auxOp = 0;
    for (let i = 0; i < ctx.KEYWORD().length; i++) {
        this.auxList.push(ctx.KEYWORD(i));
    }
    return "";
};
```

Лістинг 3.4 – Граматичний гість.

Відвідувач повинен пройти через кодовий скрипт через елементи, вказані в синтаксичному аналізаторі, і кожен аспект генерує відповідний код. Згенерований відвідувачем код – це не що інше, як рядок, інкрементований і заповнений до кінця дерева синтаксичного аналізу. Всі ключові слова також зберігаються в списку, так що в кінці повертається список і рядок, що містить згенерований скрипт. Список ключових слів необхідний, тому що його потрібно буде зіставити після генерації цього коду.

3.2.3 Контролери

Контролери відповідають за виконання системних операцій. За своєю структурою вони схожі на моделі. Для кожної моделі існує файл, який відповідає за управління процесами, пов'язаними з цією колекцією або моделлю. У кожному контролері доступно декілька операцій відповідно до потреб кожного з них, але спільними для всіх є операції створення, читання, оновлення та видалення (CRUD).

Окрім цих операцій, існують ще інші, характерні лише для деяких моделей, наприклад, виконання примітивних тестів, що є операцією, розробленою лише у контролері тестів, створення нових наборів тестів, які використовують розроблений DSL, та виконання тих самих наборів тестів, що є операціями, визначеними лише у контролері пакетів.

Наведені нижче операції відрізняються від звичайного CRUD. Однак, враховуючи контекст системи, вони є фундаментальними для її роботи:

– Метод для виконання примітивного тесту, що передає в якості аргументів директорію, де зберігаються електронні тестові драйвери, ідентифікатор тесту, який потрібно виконати, та параметри тесту.

Робиться запит для отримання всієї інформації, пов'язаної з тестом, і починається відлік часу виконання. Потім запускається драйвер, відповідальний за виконання тесту, який виконує його і повертає результати. Відлік часу виконання зупиняється, як тільки надходять результати, і час виконання тесту зберігається.

Нарешті, деяка інформація про тест додається до результатів, які зберігаються у звітах, і повертається об'єкт, що містить результати виконання тесту (лістинг 3.5):

```
module.exports.runTest = async (driversDirectory, idTest,
defaultParam) => {
    let test = await Test.findOne({ _id: idTest }).exec();
    let startTime = process.hrtime();

    exec(`${driversDirectory}\\${test.module} "${idTest}"
"${defaultParam}"`,
        (err, stdout, stderr) => {
            if (err) return stderr;
            else {
                let endTime = process.hrtime(startTime);
                let result = JSON.parse(stdout);
                result.runtime = (endTime[1] /
1000000).toFixed(3);
                result.id_test = idTest;
                result.module = test.module;
                result.name = test.name;
                return result;
            }
        });
};
```

Лістинг 3.5 – Виконати один примітивний тест.

– Метод для створення нового тестового набору і вставки його в базу даних, передаючи об'єкт з атрибутами пакета як аргумент.

Цей метод починається з використання визначеної граматики за допомогою лексера і синтаксичного аналізатора для аналізу коду скрипта, написаного користувачем.

Потім створюється дерево синтаксичного аналізу, яке генерується і передається від відвідувача граматики як аргумент. Якщо в дереві синтаксичного аналізу не знайдено жодної помилки, відвідувач пройде по цьому дереву і згенерує код для нового набору тестів.

Після генерації коду нового скрипту його буде записано у файл, який буде збережено у директорії, де знаходяться набори тестів системи.

Після цього новий набір тестів також буде додано до бази даних (лістинг 3.6):

```
module.exports.insertPackage = async (package) => {
    let chars = new antlr4.InputStream(package.script);
    let lexer = new Lexer(chars);
    let tokens = new antlr4.CommonTokenStream(lexer);
    let parser = new Parser(tokens);
    parser.buildParseTrees = true;
    let tree = parser.test();
    if (tree.parser._syntaxErrors === 0) {
        let listOfTests = await Test.getTests();
        let visitor = new Visitor(listOfTests);
        visitor.visitTest(tree);
        let textFile = visitor.getRes() + " ";
        let t = visitor.getTests();
        let tests = t.filter(function (elem, pos) {
            return t.indexOf(elem) === pos;
        });
        let fileName = package.name
            .toLowerCase()
            .replace(/\s/g, '_') + ".js";
        let filePath = scripts_path + fileName;
        fs.writeFile(filePath, textFile, "utf8", function (err) {
            if (err) throw err;
            let t = new Package({
                name: package.name,
                description: package.description,
                code: package.script,
                path: fileName,
                tests: tests,
            });
            return t.save();
        });
    } else {
        return { errors: tree.parser._syntaxErrors };
    }
};
```

Лістинг 3.6 – Створіть новий набір тестів.

– Метод для виконання набору тестів, що передає ідентифікатор набору тестів для запуску як аргумент.

Цей метод починається із запиту до колекції пакетів для отримання інформації про пакет, який потрібно виконати. Після отримання цієї інформації йому потрібно лише імпортувати файл коду тестового набору і викликати метод "run", який запускає виконання тестового набору. Наприкінці він чекає на результати і повертає їх (лістинг 3.7).

```
module.exports.runPackage = async (idPackage) => {
    let package = await Package.findOne({ _id: idPackage
}).exec();
    if (package) {
        const file = require("../../public/Packages/" +
package.path);
        return await file.execute();
    }
    return {};
};
```

Лістинг 3.7 – Виконати набір тестів.

Продемонстровані та пояснені операції є прикладами різних типів процедур, які виконує та підтримує система. Однак ми бачимо, що всі вони мають спільну рису – кожна з них виконує лише певну дію, що дозволяє нам ізолювати ці дії і часто використовувати їх у різних частинах коду для інших цілей. Це також дозволить краще підтримувати ці операції, оскільки при необхідності внести будь–які зміни в будь–яку з них, це буде зроблено лише один раз і в зазначеному місці, замість того, щоб вносити зміни в різних місцях, що в довгостроковій перспективі могло б легко призвести до неузгодженостей в коді.

3.2.4 Маршрути

Маршрути сервера, як уже згадувалося раніше, відповідають за визначення запитів, які може надіслати клієнт, і в даному випадку саме вони отримують ці запити, пересилають їх для виконання операцій, необхідних для їх задоволення, і, врешті–решт, надсилають дані клієнту. Спосіб побудови маршрутів базується на URL–адресі. З кожним запитом асоціюється URL–адреса. Оскільки визначений API відповідає архітектурі REST, ці маршрути будуть слідувати певним і точним форматам, щоб помітити тип операції, яку потрібно виконати.

Як ми бачили раніше на контролерах, CRUD–операції є найпоширенішими та найчастіше зустрічаються на маршрутах. Існує також спосіб позначати запити,

щоб визначити категорію функцій, з якими вони мають справу. В даному випадку це HTTP–запити.

У цій системі реалізовано чотири типи рекомендацій, а саме: метод get, який відповідає за отримання інформації з сервера за заданим уніфікованим ідентифікатором ресурсу (URI), запит post, який використовується для надсилання даних на сервер, запит put, який замінює всі поточні представлення цільового ресурсу завантаженим контентом, і запит delete, який відповідає за видалення всіх сучасних моделей цільового ресурсу, наданих за допомогою URI.

Всі розроблені API сервіси, доступні клієнту, ми бачимо в попередніх таблицях.

Тут не буде детально описано реалізацію всіх запитів, а лише деяких, як наочний приклад формату реалізації, який завжди однаковий, за винятком деяких більш складних запитів. Наприклад, у лістингу 3.8, наведеному нижче, показано реалізацію маршруту для запиту get/test:

```
router.get("/tests", function (req, res) {
    Tests.getTests()
        .then((data) => res.jsonp(data))
        .catch((error) => res.status(500).jsonp(error));
});
```

Лістинг 3.8 – Приклад реалізації GET–запиту.

У цьому прикладі в рядку 2 ми бачимо використання "Tests", яке є посиланням, вже імпортованим в контролер тестів. Потім з цим посиланням викликається метод "getTests", який експортується в контролері тестів. Таким чином, відбувається саме те, що було описано раніше.

Маршрутизатор перенаправляє операцію до відповідального за неї контролера. Потім він просто чекає на результати, щоб повернути їх клієнту.

У лістингу 3.9, показаному нижче, ми бачимо реалізацію маршруту для запиту пошти/пакетів:

```
router.post("/packages", function (req, res) {
    Packages.insertPackage(req.body)
        .then((data) => res.jsonp(data))
        .catch((error) => res.status(500).jsonp(error));
});
```

Лістинг 3.9 – Приклад реалізації POST–запиту.

У цьому прикладі процес дуже схожий на попередній. Єдина відмінність полягає в тому, що інформація, збережена в системі, міститься в тілі запиту ("req.body"). Тому необхідно передати цю інформацію методу, який займається операцією. У лістингу 3.10, наведеному нижче, показано реалізацію маршруту для запиту put/schedules/:idSchedule:

```
router.put("/schedules/:idSchedule", function (req, res) {
    Schedules.updateSchedule(req.params.idSchedule, req.body)
        .then((data) => res.jsonp(data))
        .catch((error) => res.status(500).jsonp(error));
});
```

Лістинг 3.10 – Приклад реалізації запиту PUT.

У цьому прикладі ми знову бачимо ту ж саму схожість, але з невеликою різницею. Запити put зазвичай передають ідентифікатор елемента, який ми хочемо оновити в підмаршруті, і, отже, щоб отримати доступ до нього, ми повинні отримати доступ до параметрів запиту ("req.params").

У лістингу 3.11, наведеному нижче, показано реалізацію маршруту для запиту delete/packages/:idPackage:

```
router.delete("/packages/:idPackage", function (req, res, next) {
    Packages.deletePackage(req.params.idPackage)
        .then((data) => res.jsonp(data))
        .catch((error) => res.status(500).jsonp(error));
});
```

Лістинг 3.11 – Приклад реалізації запиту DELETE.

У цьому прикладі ми знову бачимо ту саму структуру, але для видалення елемента з системи нам потрібно лише отримати доступ до параметрів запиту, щоб отримати ідентифікатор і передати його контролеру.

3.3 Фронтенд

Фронтенд – це рівень системи, який відповідає за створення та керування графічними інтерфейсами. Для цієї системи існує два типи користувачів. Перший тип користувачів, більш фундаментальний, матиме доступ лише до виконання примітивних тестів та наборів тестів. Другий тип користувачів, вже відповідальний за управління системою і наборами тестів, має доступ до всіх інших функціональних можливостей.

3.3.1 Компоненти

Як уже згадувалося, розробка компонентів у React стає перевагою. Проте, щоб оволодіти технологією, потрібно розуміти її основи і те, як компоненти взаємодіють між собою. Три поняття, які ми виділили тут, – це стан компонента, який є змінним і може бути змінений самим елементом. Реквізит – це інформація про стан компонента і, нарешті, події – це те, як дочірній компонент повинен інформувати батьківський компонент про зміни.

Таким чином, щоб зрозуміти, як ці поняття застосовуються на практиці і отримати максимальну користь від використання React–компонентів, ми можемо побачити нижче, на рисунку 3.1, ілюстрацію того, як ці поняття пов'язані між собою:

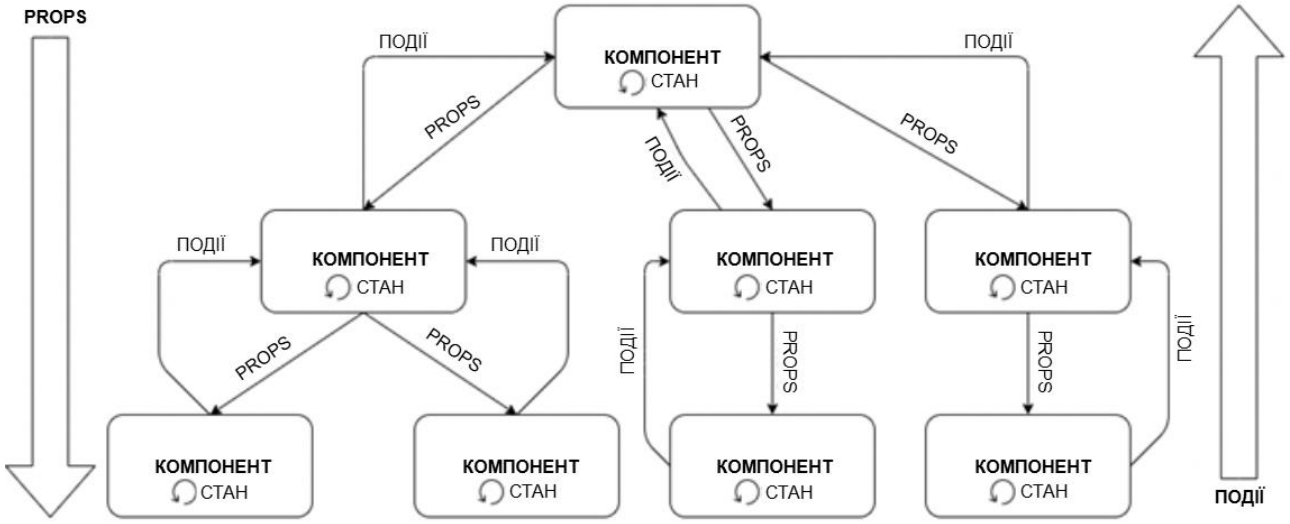

Рисунок 3.1 – Взаємодія між компонентами.

3.3.2 Отримання даних API

Ще одним важливим аспектом для того, щоб ця частина системи працювала так, як заплановано, є отримання даних, якими керує серверний рівень. Побудовані графічні інтерфейси повинні бути оптимізовані та швидкі в отриманні даних, полегшуючи таким чином роботу користувача, адже користувач не повинен довго чекати на завантаження сторінок, а дані повинні бути отримані в найкращий спосіб.

Тут було прийнято рішення, що батьківські компоненти кожної сторінки роблять запити даних до API під час її створення. Таким чином, сторінки системи працюють таким чином, що дані запитуються і завантажуються щоразу, коли користувач змінює сторінку або переходить на нову сторінку.

Спосіб отримання даних – через HTTP. Щоб зробити код зрозумілішим, було створено файл лише для запиту даних з API.

Цей файл містить базову URL–адресу API даних, а всі форми додають лише маршрут і підмаршрут. Це можна побачити нижче у лістингу 3.12:

```
export const getTests = async () => {
    try {
        const response = await axios.get(`${url}/tests`);
        return response.data;
    } catch (error) {
        const statusCode = error.response ? error.response.status
: 500;
        throw new Error(statusCode.toString());
    }
};
```

Лістинг 3.12 – Приклад запиту на отримання даних API.

У цьому прикладі показано, як здійснюються HTTP–запити до API за допомогою імпортованого модуля "Axios". Ще однією важливою особливістю, яку ми бачимо в цьому прикладі, є використання ключового слова "await", яке в цьому конкретному випадку змушує метод чекати на результати від API.

3.3.3 Інтерфейси користувача

Розмежування доступу до сторінок для кожного користувача було здійснено через логін на першій сторінці, який дозволить призначити користувачеві JSON Web Token (JWT) і надасть йому доступ лише до відповідних функціональних можливостей. Після цього користувач повинен ввести свій ідентифікатор (наданий CONTROLAR) і увійти у відповідний режим. Наприклад, якщо це оператор, він повинен увійти в режим виконання, а якщо системний менеджер, то в режим конфігурації.

Виконання примітивних тестів або тестових пакетів можливе лише на сторінці, доступній оператору. Оператор має список всіх примітивних тестів системи. Щоб запустити ці тести, він повинен вибрати потрібні і виконати їх усі одночасно. Після вибору тестів і передачі їх у конвеєр виконання, оператор повинен натиснути кнопку для запуску. Система виконає тести, і в кінці буде представлена таблиця з отриманими результатами. Інтерфейс виконання та таблиця

результатів, що показується користувачеві, можна переглянути нижче, на рисунках 3.2 та 3.3 відповідно:

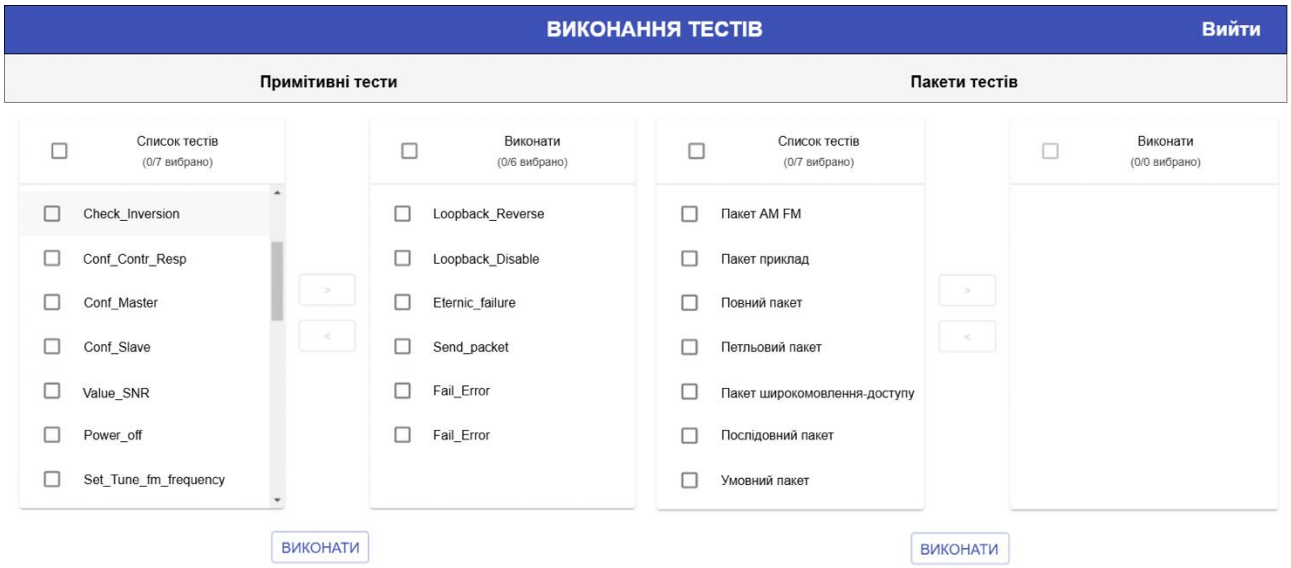

Рисунок 3.2 – Сторінка виконання

Описаний вище метод вибору та виконання примітивних тестів користувачем є аналогічним для тестових наборів.

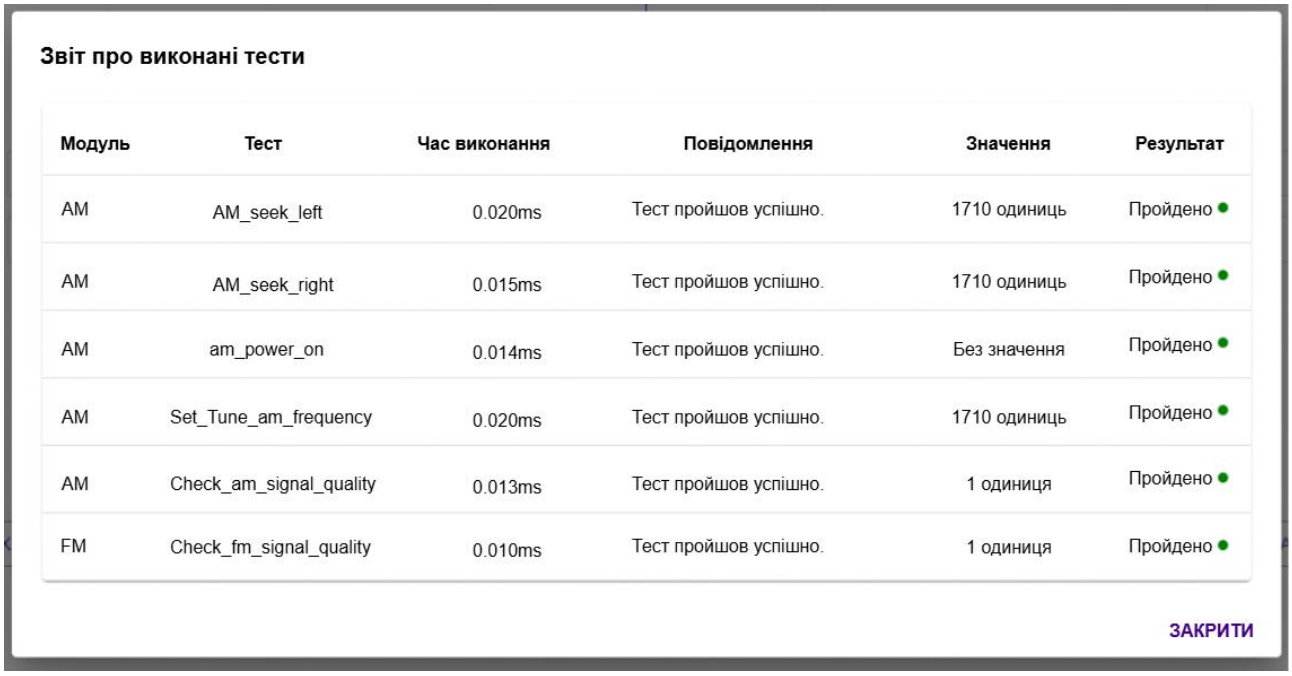

Рисунок 3.3 – Таблиця результатів виконання

Таблиця результатів показує драйвери виконання та деякі метадані, додані системою. Критичною метрикою для результатів тесту є те, чи пройшов тест, чи не пройшов, чи виявився безрезультатним. Мета цих інтерфейсів – бути максимально функціональними і простими для прийняття рішень.

Системному менеджеру або адміністратору буде доступно більше сторінок і ресурсів. Наприклад, на рисунку 3.4 показано сторінку звіту про виконання з таблицею всіх звітів про виконання, створених у системі.

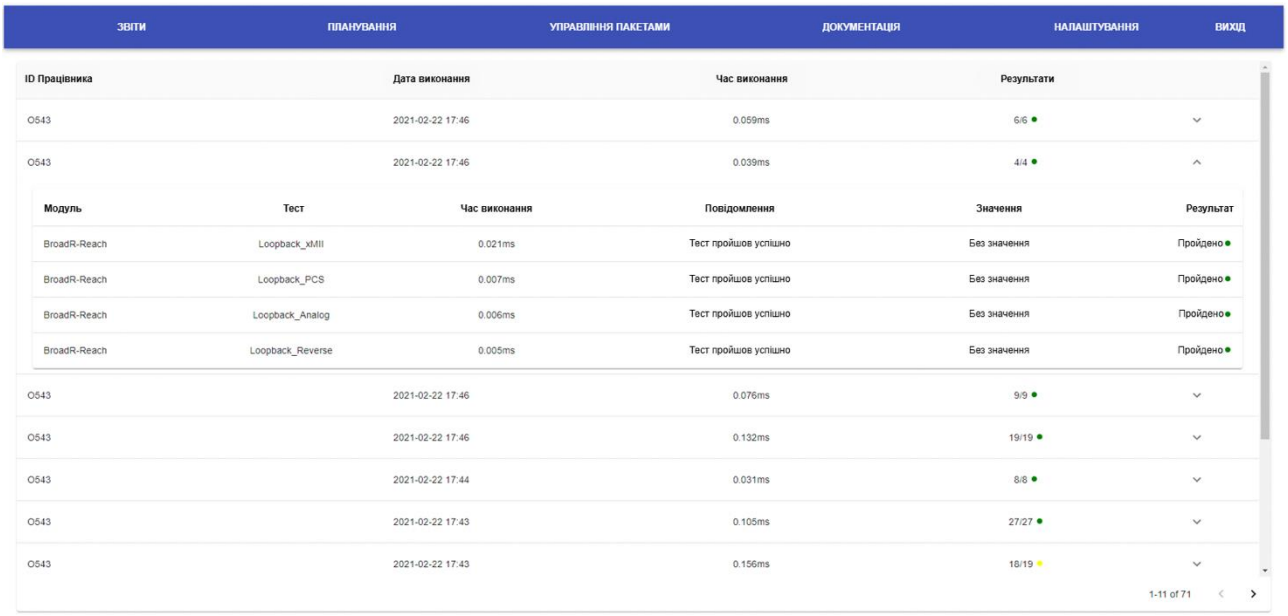

Рисунок 3.4 – Сторінка звіту

Кожен рядок таблиці дозволяє розширювати, що відкриває внутрішню таблицю, яка деталізує результати всіх тестів, виконаних у цьому звіті.

Користувачеві також доступна сторінка для керування та налаштування нових наборів тестів для системи, яку показано на рисунку 3.5. На цій сторінці користувач має у своєму розпорядженні список існуючих пакетів у системі, де він може їх видалити або відредагувати. Також є форма для створення нового тестового набору, де користувачеві потрібно лише вказати назву, опис та код нового тестового набору. Код пишеться за допомогою DSL.

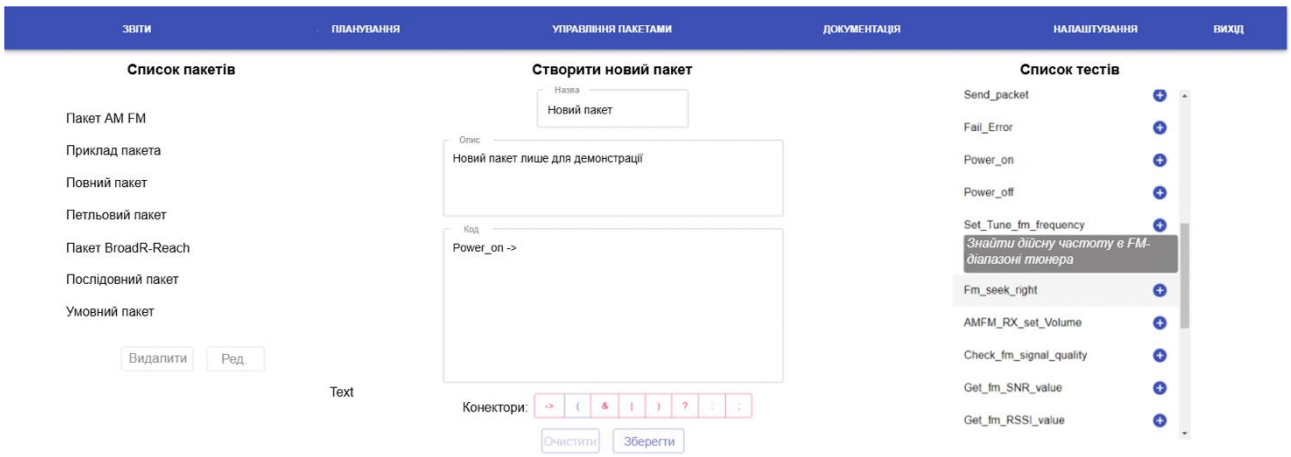

Рисунок 3.5 – Сторінка керування пакетами

У менеджера також є сторінка, на якій він має доступ і повинен оновлювати налаштування системи, коли це необхідно. Ці налаштування необхідні для роботи системи, оскільки вони використовуються у фундаментальних процесах. Налаштування включають ідентифікатор менеджера, який є єдиним, хто має доступ до цього режиму системи, директорію bin MongoDB, тобто директорію, де знаходяться виконувані файли MongoDB і яка дозволяє витягувати та імпортувати дані, директорію, де встановлена сама система, і, нарешті, директорію, куди слід експортувати резервні копії, а також звідки їх слід імпортувати. Наприклад, цю сторінку показано на малюнку 3.6:

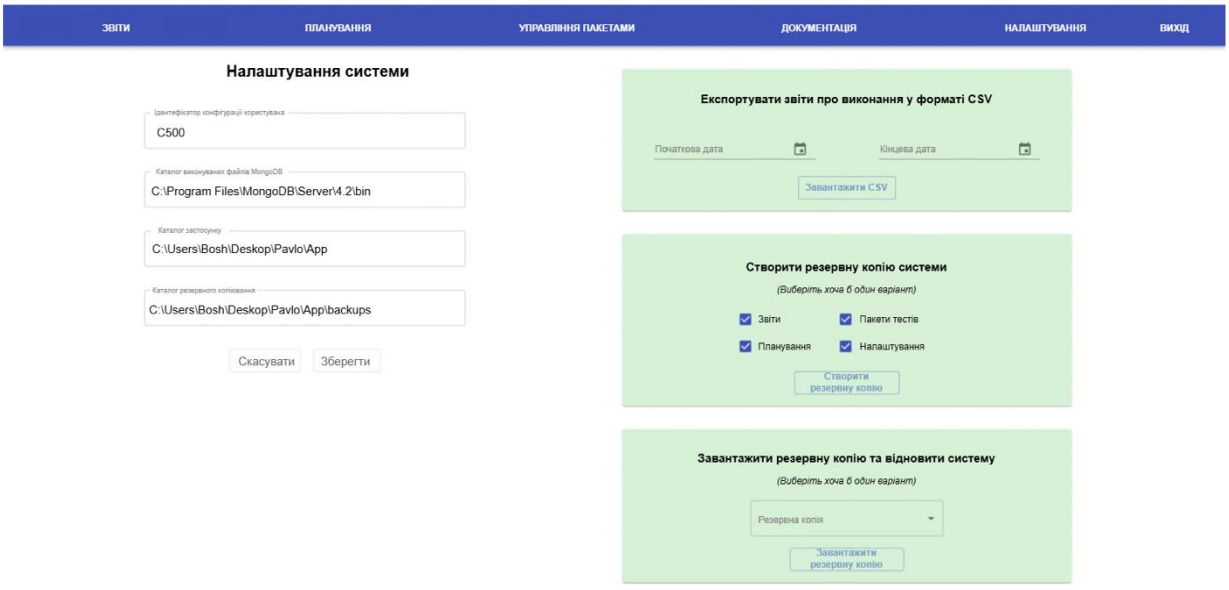

Рисунок 3.6 – Сторінка конфігурації системи та експорт і імпорт даних

На цій сторінці можна керувати конфігураціями системи, експортувати звіти про роботу системи у форматі CSV, які можна фільтрувати, створювати резервні копії, налаштовувати дані, які потрібно скопіювати, а також відновлювати резервні копії системи. Резервні копії дозволять системі постійно бути в курсі ситуацій збою або пошкодження даних. Однак ці функції вимагають, щоб конфігурації системи були правильно заповнені та коректні.

3.4 Валідація

Система була реалізована з дотриманням усіх встановлених вимог. Було створено кілька тестових кейсів для валідації рішення та підтвердження досягнення поставлених цілей. Перші тести були проведені на функціональність системи, виконання тестів та автоматизацію оновлення. Потім було змодельовано кілька тестових сценаріїв, і система поводилася як очікувалося, пройшовши всі проведені тести. Проведені тести та їх результати ми можемо побачити в Додатку А.

Ця таблиця містить рядок тестового кейсу, який описує тести, функціональність тестування та кроки тестування, які тестувальник повинен чітко виконати, щоб відтворити той самий результат або спробу. Тестові дані відповідають даним, які будуть необхідні для виконання тесту. Очікуваний результат – це ефект, якого повинен досягти тест, щоб відповідати системним вимогам. Фактичний вплив – це результат, який тест отримав після виконання і проходження/непроходження випробування.

Після виконання тестових кейсів, описаних вище, тестові кейси були виконані для всіх інших функцій системи, при цьому таблиці результатів мають однаковий формат. Ми можемо побачити результати та тестові кейси, що залишилися, у наступних Додатках Б,В, Г.

Загалом було проведено 28 тестових кейсів, які охоплювали всю функціональність системи, деякі з них містили більше одного тестового кейсу.

Більше тестових кейсів не було проведено, оскільки це зайняло б дуже багато часу, але проведені тестові кейси вважалися найбільш повними і, отже, дадуть найбільш повне покриття вимог. Проаналізувавши всі результати, отримані в тестах, і переконавшись, що всі вони пройшли, можна сказати, що всі вимоги були успішно реалізовані. В результаті система готова до інтеграції з іншими компонентами.

3.5 Висновок до третього розділу

У третьому розділі реалізовано прототип інформаційно-аналітичної підсистеми самодіагностики для кіберфізичних систем. Детально описано реалізацію бази даних на основі MongoDB, з визначенням ключових колекцій для зберігання конфігурацій, тестів, звітів, розкладів та тестових пакетів. Побудовано серверну частину системи з використанням Node.js і Express, дотримуючись архітектурного шаблону MVC.

Отримані результати демонструють працездатність запропонованого підходу до самодіагностики в реальному часі та підтверджують його придатність для інтеграції в КФС з метою підвищення надійності та оперативності діагностичних процесів.

Були запропоновані різні рівні системи, а також технології, методи і стратегії, використані для розробки плану, який би найкращим чином відповідав усім вимогам. Весь код не пояснювався і не деталізувався, оскільки він є наскрізним, але найбільш релевантні частини були пояснені, що дозволило тим, хто читав його, відтворити цю роботу і застосувати її в своєму контексті.

Слід також зазначити, що розроблена система готова до інтеграції з КФС і з гарантованою інтеграцією електронних тестів і, отже, готова до проведення самодіагностики машин CONTROLAR. Для перевірки реалізації системи та її відповідності встановленим вимогам було проведено 28 тестових кейсів, щоб

покрити всі потреби. Результати показують, що всі тестові кейси були схвалені і, отже, система відповідає всім запропонованим вимогам.

РОЗДІЛ 4. БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

4.1 Питання щодо безпеки життєдіяльності

Безпека життєдіяльності у процесі розробки, впровадження та експлуатації інформаційно-аналітичної підсистеми для самодіагностики в кіберфізичних системах (КФС) є важливим аспектом, що безпосередньо впливає на захист життя і здоров’я працівників, безперебійну роботу обладнання та зниження ризику виникнення аварійних ситуацій.

Фізична безпека персоналу: При розміщенні та експлуатації кіберфізичних систем, до складу яких можуть входити електронні модулі, високовольтні компоненти, сенсорні системи та рухомі механізми, необхідно дотримуватись вимог Закону України «Про охорону праці» (від 14.10.1992 № 2694-XII), який регламентує безпечні умови праці та зобов’язує роботодавця створити умови, що унеможливлюють вплив шкідливих та небезпечних факторів [49].

Зокрема, при розробці і тестуванні системи потрібно:

– забезпечити електробезпеку згідно з Правилами безпечної експлуатації електроустановок споживачів (ПБЕЕС);

– дотримуватися вимог Державних санітарних норм і правил при роботі з ПЕОМ (ДСанПіН 3.3.2.007-98) щодо тривалості роботи за комп’ютером, ергономіки робочого місця та рівня освітлення;

– проводити навчання та перевірку знань з питань БЖД для осіб, які мають доступ до технічних засобів автоматизації, згідно з Типовим положенням про порядок проведення навчання і перевірки знань з питань охорони праці.

Електробезпека та захист від ураження: Під час взаємодії з тестовими модулями, які можуть бути під напругою, передбачено виконання робіт відповідно до ГОСТ 12.1.019-79 «Електробезпека. Загальні вимоги і номенклатура видів захисту». Всі елементи тестового обладнання мають бути заземлені, а корпуси – захищені від дотику до струмопровідних частин.

Роботи з налагодження електронних компонентів слід виконувати лише з використанням засобів індивідуального захисту (ЗІЗ): діелектричних рукавиць, ковриків, інструменту з ізольованими ручками. Перед запуском тестів – обов’язкова візуальна перевірка цілісності з’єднань та стану пристроїв.

Вимоги до приміщення та організації робочого місця: Сервери, тестові пристрої та моніторингові системи повинні бути розміщені у приміщеннях, які відповідають ДБН В.2.2-3:2018 «Будинки і споруди. Заклади освіти» (у частині вимог до електронного обладнання) або відповідним профільним стандартам. Робоче місце оператора повинно забезпечувати:

– нормоване мікрокліматичне середовище (температура 20–24°С, вологість 40–60%);

– оптимальний рівень освітлення (не менше 300 лк)

– обмеження впливу шуму та вібрацій відповідно до ДСН 3.3.6.037–99.

Інформаційна безпека як частина БЖД: Сучасні КФС оперують даними, зокрема параметрами роботи обладнання та результатами тестування. Несанкціонований доступ до таких даних може спричинити не лише порушення цілісності системи, але й призвести до аварійної ситуації на виробництві. Тому, згідно з Законом України «Про інформацію», необхідно впроваджувати політику інформаційної безпеки, яка включає:

– автентифікацію та авторизацію користувачів;

– обмеження доступу до адміністративних функцій;

– аудит дій користувачів;

– використання зашифрованих з’єднань для передачі даних.

Надзвичайні ситуації та дії при аваріях

Згідно з вимогами Кодексу цивільного захисту України та ДСТУ ISO 45001:2019 «Системи управління охороною праці та безпекою здоров’я», підприємство, що впроваджує КФС з функцією самодіагностики, повинно:

– мати інструкції на випадок виникнення аварій;

– обладнати приміщення системами пожежної безпеки;

– навчити персонал реагуванню на інциденти...

4.2 Питання з основ охорони праці

Організація безпечних умов праці під час розробки, тестування та впровадження інформаційно-аналітичної підсистеми самодіагностики в кіберфізичних системах ґрунтується на вимогах чинного законодавства України та відповідних нормативно-правових актів.

Загальні вимоги охорони праці: Основні положення охорони праці визначено Законом України «Про охорону праці», який зобов'язує роботодавця створити безпечні умови праці на всіх етапах виробничого процесу та провести відповідне навчання персоналу. Згідно з вимогами цього закону:

– кожен працівник має пройти інструктаж з охорони праці (вступний, первинний, періодичний);

– робоче місце має бути безпечним і відповідати Державним санітарним правилам і нормам ДСанПіН 3.3.2.007–98;

– обладнання повинно регулярно перевірятись і допускатись до експлуатації лише за наявності технічної документації та інструкцій.

Охорона праці під час роботи з електронними та комп'ютерними системами: Розробка й експлуатація підсистеми самодіагностики передбачає роботу з електричними приладами, периферійним обладнанням, мережевими пристроями та програмними серверами. Це зумовлює потребу дотримання:

– Правил технічної експлуатації електроустановок споживачів (ПТЕЕС);

– Правил безпечної роботи з електроустаткуванням;

– Інструкцій з безпечного користування ПЕОМ, які регулюють час роботи за комп'ютером, відстань до монітора, положення клавіатури та сидіння працівника. nПри роботі в умовах потенційного електромагнітного випромінювання або з високочастотними сигналами (у складі КФС) необхідно враховувати обмеження, передбачені Нормами допустимого рівня випромінювання (ДСП 9.2.6.082–2001), та виконувати заземлення всього обладнання. Вимоги до організації робочого

місця: Згідно з ДБН В.2.2-40:2018, робоче місце розробника або тестувальника повинно:

– бути обладнане регульованим стільцем та столом;

– забезпечувати природне або комбіноване освітлення з мінімальним рівнем 300 лк;

– бути розташоване на достатній відстані від джерел шуму та вібрацій.

Рекомендується дотримуватися режиму праці: не більше 6 годин роботи за монітором щоденно, з перервами кожні 1–2 години тривалістю не менше 10–15 хвилин, згідно з наказом МОЗ № 125 від 10.12.1996.

Пожежна безпека: Під час роботи із серверними системами та електронними пристроями необхідно враховувати Правила пожежної безпеки в Україні (НАПБ А.01.001–2014). Системне обладнання має розміщуватись з урахуванням:

– доступу до засобів пожежогасіння (вогнегасники типу ВВК-2, ВП-5);

– відсутності займистих матеріалів поблизу блоків живлення;

– наявності евакуаційних шляхів, передбачених ДБН В.1.1-7:2016.

Медичне обслуговування та профілактика: Працівники, які безпосередньо взаємодіють із кіберфізичними системами, повинні проходити:

– попередній медичний огляд (при прийомі на роботу);

– періодичні медичні огляди згідно з Порядком проведення медичних оглядів працівників певних категорій (наказ МОЗ № 246).

Це дозволяє своєчасно виявляти професійні захворювання, знижувати рівень професійних ризиків і зберігати працездатність персоналу [50].

4.3 Висновок до четвертого розділу

У межах цього розділу розглянуто ключові питання, пов’язані із забезпеченням безпеки життєдіяльності та дотриманням вимог охорони праці під час розробки та експлуатації інформаційно-аналітичної підсистеми

самодіагностики в кіберфізичних системах. Було проаналізовано потенційні ризики для здоров’я та життя працівників, які працюють з електронним, комп’ютерним та тестовим обладнанням, а також запропоновано заходи щодо їх попередження.

Особливу увагу приділено фізичній безпеці персоналу, зокрема електробезпеці при взаємодії з високовольтними системами, необхідності заземлення пристроїв, використанню засобів індивідуального захисту та дотриманню технічної документації. Враховано нормативні документи, такі як Закон України «Про охорону праці», Правила безпечної експлуатації електроустановок споживачів, ДСанПіН 3.3.2.007-98 та інші.

Окремо акцентовано увагу на інформаційній безпеці як складовій загальної безпеки життєдіяльності – через необхідність впровадження механізмів автентифікації, обмеження доступу до критичних компонентів та протидії несанкціонованим діям.

У підрозділі, присвяченому основам охорони праці, детально розглянуто вимоги до організації робочого місця, санітарно-гігієнічні умови, пожежну безпеку, організацію інструктажів, наявність медичних оглядів і порядок поводження у разі аварійних ситуацій. Наведено посилання на відповідні національні стандарти, санітарні норми та типові положення.

Загалом, дотримання зазначених вимог та рекомендацій сприяє формуванню безпечного виробничого середовища, запобіганню професійним ризикам і забезпечує ефективну роботу персоналу при розгортанні та супроводі КФС у промислових умовах..

ВИСНОВКИ

Під час виконання даної роботи дійшли висновку, що основним внеском у розвиток проекту є розробка архітектури для інтеграції системи самодіагностичного тестування в ЦСК та її реалізація. Інтегрований підхід дозволить організації самодіагностувати свої машини в режимі реального часу, забезпечуючи таким чином їхню цілісність.

На основі сучасного аналізу ми виявили, що існуючі рішення для систем тестування все ще зосереджені лише на тестуванні програмного забезпечення і дуже мало уваги приділяють інтеграції інших категорій тестів, а отже, їх інтеграції в КФС. Однак у світлі знань, отриманих в результаті аналізу систем тестування програмного забезпечення та автоматизації тестування, можна створити план з деякими з цих характеристик, який призначений для інтеграції та самодіагностики КФС. З цією метою було розроблено архітектуру системи самодіагностики тестів, яка поєднує методологію KDT з DSL для управління та конфігурації тестів системи. Ця архітектура забезпечує модульне і масштабоване рішення для інтеграції системи з КФС і виконання будь–якого тесту. Крім того, була розроблена інша архітектура для розширення та інтеграції системи самотестування в КФС, яка доводить модульність запропонованої архітектури для системи самотестування, демонструючи, як ми можемо розвивати її в КФС.

Запропонована модульна і розширювана архітектура являє собою інновацію для систем самодіагностики і досліджень КПС. Вона дозволяє об'єднати ці два плани, використовуючи методологію KDT з DSL для управління та конфігурації системних тестів. Крім того, ця архітектура дозволяє запускати тести віддалено або будь–якою іншою системою з дозволом на запит HTTP до REST API. Хоча архітектура орієнтована на застосування КФС, вона також може бути застосована до будь–якої системи, оскільки є універсальною для прийняття будь–якого тесту. Цією роботою ми довели, що можна інтегрувати системи самодіагностики в КФС

за допомогою практичного та універсального рішення, що поєднується з іншими тестовими системами.

Реалізація системи відповідно до заданих вимог на основі запропонованої архітектури доводить, що система може бути модульною і дозволяти самодіагностику шляхом проходження будь–якого тесту. Для перевірки реалізації системи та її відповідності встановленим вимогам було виконано 28 тестових кейсів, які охоплюють всі вимоги. Результати показують, що всі тестові кейси пройдені, тобто система відповідає всім запропонованим вимогам. Таким чином, можна побачити, що внески гарантують безпеку, продуктивність і функціональність при використанні машин CONTROLAR, і тепер їх можна діагностувати в режимі реального часу, що дозволяє таким клієнтам, як Bosch, максимально ефективно використовувати їх у виробничому середовищі.

В першому розділі кваліфікаційної роботи освітнього рівня «Бакалавр»:

– Подано загальну характеристику кіберфізичних систем, визначено їх роль у сучасному виробництві та необхідність впровадження самодіагностики;

– Розглянуто основні проблеми, пов’язані з функціональністю тестових систем, зокрема труднощі виявлення збоїв без належних інструментів діагностики;

– Висвітлено сучасні підходи до автоматизованого тестування, включаючи методології KDT, предметно-орієнтовані мови (DSL), синтаксичний аналіз з використанням ANTLR, а також REST-архітектуру як технологічну основу для побудови клієнт-серверної взаємодії;

– Проаналізовано стан концепції КФС у наукових дослідженнях, проблематику виявлення збоїв, а також актуальні інструменти й технології, які можуть бути використані для створення гнучкої системи самодіагностики.

В другому розділі кваліфікаційної роботи:

– Досліджено вимоги до побудови інформаційно-аналітичної підсистеми самодіагностики в КФС, зокрема особливості клієнт-серверної архітектури, структури даних та взаємодії компонентів системи;

– Обґрунтовано вибір технологій для реалізації системи, таких як Node.js, React.js, MongoDB, а також підходів до автоматизації тестування на основі KDT та DSL;

– Сформовано логічну і технічну архітектуру підсистеми, включаючи моделі управління тестами, конфігурацією сценаріїв та інтеграцію з КФС, що забезпечує гнучкість, масштабованість і самодіагностику в режимі реального часу.

В третьому розділі кваліфікаційної роботи:

– Розроблено повнофункціональний прототип інформаційно-аналітичної підсистеми самодіагностики на базі технологій Node.js, Express, MongoDB та React.js;

– Запропоновано механізм створення та виконання тестових сценаріїв із використанням предметно-орієнтованої мови (DSL), що поєднується з методологією тестування на основі ключових слів (KDT);

– Спроєктовано базу даних із відповідною структурою колекцій для зберігання конфігурацій, тестів, звітів, розкладів і пакетів, а також граматику та лексичний аналізатор для обробки сценаріїв самодіагностики;

– Протестовано ключові функції системи, включаючи запуск примітивних тестів, створення тестових пакетів та генерацію звітів, що підтвердило її працездатність і відповідність поставленим вимогам.…

У розділі «Безпека життєдіяльності, основи охорони праці» розглянуто нормативно-правову базу та заходи, що забезпечують безпечні умови праці під час розробки та експлуатації інформаційно-аналітичної підсистеми в кіберфізичних системах.

Висвітлено вимоги до електробезпеки, ергономіки робочого місця, пожежної безпеки, а також організації інструктажів, медичних оглядів і профілактичних заходів відповідно до чинного законодавства України та галузевих стандартів. Особливу увагу приділено інформаційній безпеці як складовій загальної безпеки життєдіяльності в умовах цифрового виробництва.

ПЕРЕЛІК ДЖЕРЕЛ

1 Lee, E.A. Cyber physical systems: Design challenges. In Proceedings of the 11th IEEE Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing, ISORC 2008, Orlando, FL, USA, 5–7 May 2008.

2 Pereira, R.B.; Brito, M.A.; Machado, R.J. Architecture Based on Keyword Driven Testing with Domain Specific Language for a Testing System; Springer International Publishing: Berlin/Heidelberg, Germany, 2020; pp. 310–316.

3 Pereira, R.B.; Ramalho, J.C.; Brito, M.A. Development of self-diagnosis tests system using a DSL for creating new test suites for integration in a cyber-physical system. Open Access Ser. Inform. 2021, 94, 1–16.

4 Leitão, P. Agent-based distributed manufacturing control: A state-of-the-art survey. Eng. Appl. Artif. Intell. 2009, 22, 979–991.

5 Minho, U.D. Informação Institucional. 2022. Ресурс: https://www.uminho.pt/PT/uminho/Informacao-Institucional/ Paginas/default.aspx (Доступно Березень 2025).

6 Algoritmi, C. Ongoing Projects, 2020. Ресурс: https://algoritmi.uminho.pt/projects/ongoing-projects/ (Доступно Березень 2025).

7 CCG. TSIM—Test System Intelligent Machines. 2020. Ресурс: https://www.ccg.pt/my-product/tsim-test-system-intelligent-machines/ (Доступно Березень 2025).

8 Controlar. Máquina Inteligente de Sistema de Testes Funcionais|Controlar. 2020. Ресурс: https://controlar.com/areas-de-negocio/sistemas-de-teste/tsim/ (Доступно Березень 2025).

9 Seshia, S.A.; Hu, S.; Li, W.; Zhu, Q. Design Automation of Cyber-Physical Systems: Challenges, Advances, and Opportunities. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. 2017, 36, 1421–1434.

10 Ngu, A.H.; Gutierrez, M.; Metsis, V.; Nepal, S.; Sheng, Q.Z. IoT Middleware: A Survey on Issues and Enabling Technologies. IEEE Internet Things J. 2017, 4, 1–20. https://doi.org/10.1109/JIOT.2016.2615180.

11 Vyatkin, V. Software engineering in industrial automation: State-of-the-art review. IEEE Trans. Ind. Inform. 2013, 9, 1234–1249.

12 Chen, Y.; Kar, S.; Moura, J.M.F. Resilient Distributed Estimation: Sensor Attacks. IEEE Trans. Autom. Control 2019, 64, 3772–3779.

13 An, L.; Yang, G.H. Enhancement of opacity for distributed state estimation in cyber–physical systems. Automatica 2022, 136, 110087.

14 Cintuglu, M.H.; Mohammed, O.A.; Akkaya, K.; Uluagac, A.S. A Survey on Smart Grid Cyber-Physical System Testbeds. IEEE тCommun. Surv. Tutor. 2017, 19, 446–464. https://doi.org/10.1109/COMST.2016.2627399.

15 Asadollah, S.A.; Inam, R.; Hansson, H. A survey on testing for cyber physical system. In Proceedings of the 27th IFIP WG 6.1 International Conference, ICTSS 2015, Sharjah and Dubai, United Arab Emirates, 23–25 November 2015; Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics).

16 Zhou, X.; Gou, X.; Huang, T.; Yang, S. Review on Testing of Cyber Physical Systems: Methods and Testbeds. IEEE Access 2018, 6, 52179–52194.

17 Carvalho, M.F.A. Automatização de Testes de Software Dashboard QMSanalyser; Technical Report; Instituto Politecnico de Coimbra: Coimbra, Portugal, 2010. Computers 2022, 11, 131 29 of 30

18 Guru99. Automation Testing Tutorial: What is Automated Testing? 2021. Ресурс: https://www.guru99.com/ automation-testing.html (Доступно Березень 2025).

19 Saufi, S.R.; Ahmad, Z.A.B.; Leong, M.S.; Lim, M.H. Challenges and opportunities of deep learning models for machinery fault detection and diagnosis: A review. IEEE Access 2019, 7, 122644–122662.

20 Shi, Z.; O'Brien, W. Development and implementation of automated fault detection and diagnostics for building systems: A review. Autom. Constr. 2019, 104, 215–229. https://doi.org/10.1016/J.AUTCON.2019.04.002.

21 Tang, J.; Cao, X.; Ma, A. Towards adaptive framework of keyword driven automation testing. In Proceedings of the IEEE International Conference on Automation and Logistics, ICAL 2008, Qingdao, China, 1–3 September 2008; pp. 1631–1636. https://doi.org/10.1109/ICAL.2008.4636415.

22 Hametner, R.; Winkler, D.; Zoitl, A. Agile testing concepts based on keyword-driven testing for industrial automation systems. In Proceedings of the IECON 2012—38th Annual Conference on IEEE Industrial Electronics Society, Montreal, QC, Canada, 25–28 October 2012; pp. 3727–3732.

23 Razak, R.A.; Fahrurazi, F.R. Agile testing with Selenium. In Proceedings of the 2011 5th Malaysian Conference in Software Engineering, MySEC 2011, Johor Bahru, Malaysia, 13–14 December 2011; pp. 217–219.

24 Tutorials Point. QTP Tutorial—Tutorialspoint. 2021. Ресурс: https://www.tutorialspoint.com/qtp/index.htm (Доступно Березень 2025)

25 Lalwani, T. QuickTest Professional Unplugged, 2nd ed.; KnowledgeInbox: 2011.

26 Kaur, M.; Kumari, R. Comparative Study of Automated Testing Tools: TestComplete and QuickTest Pro. Int. J. Comput. Appl. 2011, 24, 1–7.

27 Focus, M. Silk Test Automation for Web, Mobile and Enterprise Apps. 2021. Ресурс: https://www.microfocus.com/enus/products/silk-test/overview (Доступно Березень 2025).

28 Lima, T.; Dantas, A.; Vasconcelos, L. Usando o SilkTest para automatizar testes: Um Relato de Experiência. In Proceedings of the 6th Brazilian Workshop on Systematic and Automated Software Testing, Natal, RN, Brazil, 23 September 2012.

29 Ranorex. Test Automation Tools|Ranorex Automated Software Testing. 2021. Ресурс: https://www.ranorex.com/testautomation-tools/ (Доступно Березень 2025).

30 Jian-Ping, L.; Juan-Juan, L.; Dong-Long, W. Application analysis of automated testing framework based on robot. In Proceedings of the International Conference on Networking and Distributed Computing, ICNDC, Hangzhou, China, 21–24 October 2012, pp. 194–197. https://doi.org/10.1109/ICNDC.2012.53.

31 Hermans, F.; Pinzger, M.; Van Deursen, A. Domain-specific languages in practice: A user study on the success factors. In Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics); Springer: Berlin/Heidelberg, Germany, 2009; pp. 423–437.

32 Ciraci, S.; Fuller, J.C.; Daily, J.; Makhmalbaf, A.; Callahan, D. A runtime verification framework for control system simulation. In Proceedings of the 2014 IEEE 38th Annual Computer Software and Applications Conference, Vasteras, Sweden, 21–25 July 2014; pp. 75–84. https://doi.org/10.1109/COMPSAC.2014.14.

33 Krueger, C.W. Software Reuse. ACM Comput. Surv. (CSUR) 1992, 24, 131–183. https://doi.org/10.1145/130844.130856.

34 Parr, T.J.; Quong, R.W. ANTLR : A Predicated- LL (k) Parser Generator. Softw. Pract. Exp. 1995, 25, 789–810. https://doi.org/10.1002/spe.4380250705.

35 Tomassetti, G. The ANTLR Mega Tutorial. 2021. Available online: https://tomassetti.me/antlr-mega-tutorial/ (accessed on 20 July 2021).

36 Parr, T.; Fisher, K. LL(*): The foundation of the ANTLR parser generator. ACM Sigplan Not. 2011, 46, 425–436. https://doi.org/10.1145/1993316.1993548.

37 Parr, T.; Harwell, S.; Fisher, K. Adaptive LL(*) Parsing: The Power of Dynamic Analysis. ACM Sigplan Not. 2014, 49, 579–598.

38 Palsberg, J.; Jay, C.B. The Essence of the Visitor Pattern. In Proceedings of the International Computer Software and Applications Conference, Washington, DC, USA, 19–21 August 1998. https://doi.org/10.1109/CMPSAC.1998.716629.

39 Cademy, C. What is REST?|Codecademy. 2021. Ресурс: https://www.codecademy.com/articles/what-is-rest (Доступно Березень 2025).

40 Costa, B.; Pires, P.F.; Delicato, F.C.; Merson, P. Evaluating a Representational State Transfer (REST) architecture: What is the impact of REST in my architecture? In Proceedings of the Working IEEE/IFIP Conference on Software Architecture 2014, WICSA 2014, Sydney, Australia, 7–11 April 2014; pp. 105–114. https://doi.org/10.1109/WICSA.2014.29.

41 Costa, B.; Pires, P.F.; Delicato, F.C.; Merson, P. Evaluating REST architectures - Approach, tooling and guidelines. J. Syst. Softw. 2016, 112, 156–180.

42 Subramanian, V. Pro MERN Stack; Apress: New York, NY, USA, 2019.

43 Inc, F. React—A JavaScript library for building user interfaces. 2021. Ресурс: https://reactjs.org/ (Доступно Березень 2025).

44 Porter, P.; Yang, S.; Xi, X. The Design and Implementation of a RESTful IoT Service Using the MERN Stack. In Proceedings of the 2019 IEEE 16th International Conference on Mobile Ad Hoc and Smart Systems Workshops, MASSW 2019, Monterey, CA, USA, 4–7 November 2019; pp. 140–145.

45 Aggarwal, S. Modern Web-Development using ReactJS. Int. J. Recent Res. Asp. 2018, 5, 133–137. Computers 2022, 11, 131 30 of 30

46 Javeed, A. Performance Optimization Techniques for ReactJS. In Proceedings of the 2019 3rd IEEE International Conference on Electrical, Computer and Communication Technologies, ICECCT 2019, Coimbatore, India, 20–22 February 2019; pp. 1–5. https://doi.org/10.1109/ICECCT.2019.8869134.

47 Obinna, E. Use the React Profiler for Performance. 2018. Ресурс: https://www.digitalocean.com/community (Доступно Березень 2025).

48 Oliveira, D.F.; Brito, M.A. Position Paper: Quality Assurance in Deep Learning Systems; SciTePress: Setúbal, Portugal, 2022; pp. 203–210. https://doi.org/10.5220/0011107100003269.

49 Гурик, О. Я., Окіпний, І. Б. (2021). Методичні вказівки для написання розділу «Безпека життєдіяльності, основи охорони праці» в кваліфікаційних роботах здобувачів освітнього рівня «бакалавр». Тернопіль: ТНТУ імені Івана Пулюя.

50 Вовк, Ю. Я., & Вовк, І. П. (2019). Охорона праці в галузі: навчальний посібник. Тернопіль: ТНТУ імені Івана Пулюя.

ДОДАТКИ

Додаток А

Результати тестових кейсів, виконаних на виконання та управління тестами.

Тестовий приклад	Етапи тестування	Дані тестування	Очікувані результати	Фактичні результати	Пройдено/ Не пройдено
Перевірте виконання примітивних тестів	1. Виберіть тести для виконання 2. Натисніть кнопку для запуску 3. Коли з'явиться вікно для підтвердження, виберіть так	Вибрані тести: am_power_on set_quotency am_seek_left	Повинна з'явитися таблиця з результатами тесту	Представлена таблиця з результатами	Пройдено
Перевірте виконання всіх примітивних тестів	1. Виберіть всі тести для виконання 2. Натисніть кнопку для запуску 3. Коли з'явиться вікно для підтвердження, виберіть так	Вибрані тести: всі доступні	Повинна з'явитися таблиця з усіма результатами тестування	Таблиця з усіма результатами	Пройдено
Перевірте наявність повідомлення, якщо не вибрано жодного тесту для сповіщення	1. Натисніть кнопку, щоб виконати без вибору тестів	Ні.	Повинно з'явитися попереджувальне повідомлення про те, що користувач не вибрав жодного тесту	З'явилося повідомлення	Пройдено
Перевірте, чи оновлює система видалені дані примітивні тести	1. Перейдіть до каталогу електронних тестових драйверів 2. Видаліть файл driverA.json з каталогу 3. Відкрийте систему в режимі виконання 4. Перевірте, чи доступні тести драйвера А	Ні.	Тести з драйвера А не повинні відображатися як вибрані	Тести не доступні	Пройдено
Перевірте, чи система оновлює додані примітивні тести	1. Перейдіть до каталогу електронних тестових драйверів 2. Додайте driverA.json з каталогу 3. Відкрийте систему в режимі виконання 4. Перевірте, чи доступні тести драйвера А	Ні.	Тести з драйвера А мають бути вибрані	Доступні тести	Пройдено
Перевірте виконання набору тестів	1. Виберіть набір тестів для виконання 2. Натисніть кнопку, щоб запустити	Відібрано тестові набори: Пакоте AM FM	Повинна з'явитися таблиця з результатами набору тестів	Представлена таблиця з результатами	Пройдено
Перевірте виконання всіх наборів тестів	1. Виберіть усі набори тестів для виконання 2. Натисніть кнопку для запуску 3. Коли з'явиться вікно для підтвердження, виберіть так	Вибрані тестові набори: всі доступні	Повинна з'явитися таблиця з усіма результатами набору тестів	Таблиця з усіма результатами	Пройдено
Перевірте наявність повідомлення, якщо немає тестового набору вибрано для сповіщення	1. Натисніть кнопку, щоб виконати без вибору тестів	Ні.	Повинно з'явитися попереджувальне повідомлення про те, що користувач не вибрав жодного тестові набори	З'явилося повідомлення з повідомлення	Пройдено

Додаток Б

Результати тестових кейсів, виконаних за звітами візуалізації, документацією примітивних тестів та плануванням виконання.

Тестовий приклад	Етапи тестування	Дані тестування	Очікувані результати	Фактичні результати	Пройдено/Не пройдено
Перевірте виконання звіти про результати діяльності	1. Відкрийте вкладку "Relatórios"	Немає	Повинна з'явитися таблиця з усіма звітами	Представлена таблиця	Пройдено.
Перевірте деталі звіту про виконання	1.Відкрити вкладку "Relatórios" 2 Виберіть звіт і натисніть на нього	Немає	Необхідно представити таблицю з доказами результатів випробувань, проведених у цьому звіті	Представлена таблиця	Пройдено.
Перевірте документацію примітиву тести	1. Відкрийте вкладку "Documentação"	Немає	Повинна бути представлена таблиця з усіма метаданими примітивного тесту	Представлена таблиця	Пройдено.
Перевірте графік роботи нового виконавця. tion	1. Натисніть на кнопку, щоб додати розклад 2. Введіть час 3. Діяти вати кнопку вибору 4. Виберіть примітивні тести для виконання 5. Виберіть набори тестів для виконання 6. Натисніть на кнопку збереження	Час: 8:00 Активний: Правда. Вибрані тести: power_on set_fm_frequency fm_seek_right	Необхідно додати новий розклад до системи	Створено розклад	Пройдено.
Перевірте планування нового виконання без вибору будь-який тест	1. Натисніть на кнопку, щоб додати розклад 2. Введіть час 3. Активуйте кнопку вибору 4. Натисніть на кнопку збереження	Час: 10:00 Активний: Правда.	Повинно з'явитися повідомлення про помилку, в якому зазначено, що має бути обрано принаймні один тест	З'явилося повідомлення	Пройдено.
Перевірте оновлення розкладу	1. Натисніть на розклад на 8:00 2. Змініть час на 9:00 3. Натисніть на кнопку "Зберегти	Час: 9:00	Розклад повинен бути оновлений	Створено розклад	Пройдено.
Перевірте видалення розкладу	1. Натисніть на кнопку опції, щоб видалити розклад, приуроченний до 9:00	Ні.	Графік повинен бути видалений	Створено розклад	Пройдено.

Додаток В

Результати тестових кейсів, виконаних в управлінні та створенні наборів тестів та експорті звітів у CSV.

Тестовий приклад	Етапи тестування	Дані тестування	Очікувані результати	Фактичні результати	Пройдено/Не пройдено
Перевірте створення нового тестового набору з неправильним скриптом	1. Відкрийте вкладку "Gestão de Pacotes" 2. Введіть назву пакунка 3. Введіть опис пакета 4. Напишіть скрипт 5. Натисніть на кнопку збереження	Назва пакета: Novo Pacote Package De-скрипт: Pacote para demonstrar erro Package Script: InventedTest -> am_power_on->	Користувачеві повинно з'явитися повідомлення про помилку з таким текстом код не є коректним	Показано попередження	Пройдено.
Перевірте створення нового набору тестів	1. Відкрийте вкладку "Gestão de Pacotes" 2. Введіть назву пакунка 3. Введіть опис пакета 4. Напишіть скрипт 5. Натисніть на кнопку збереження	Назва пакунка: Новий пакунок Опис пакунка: Демонстраційний пакунок Скрипт пакунка: power_on -> am_power_on;	Набір тестів має бути створений і повинен з'явитися у списку зліва	Створено і доступний набір тестів	Пройдено.
Перевірте оновлення тестового набору	1. Відкрийте вкладку "Gestão de Pacotes" 2. Натисніть на пакет з назвою "Новий пакет" 3. Натисніть на кнопку редагування 4. Змініть назву 5. Натисніть на кнопку збереження	Пакет ім'я: Новий Видалення пакунка	Необхідно оновити тестовий набір	Оновлений набір тестів	Пройдено.
Перевірте видалення тестового набору	1. Відкрийте вкладку "Gestão de Pacotes" 2. Натисніть на пакунок з назвою "Новий пакунок для видалення" 3. Натисніть на кнопку видалення	Hi.	Тестовий набір має бути видалено	Видалено набір тестів	Пройдено.
Перевірте експорт звітів у CSV з датами, що не охоплюють ered	1. Відкрийте вкладку "Configurações" 2. Введіть дату початку експорту CSV 3. Введіть дату закінчення експорту CSV 4. Натисніть кнопку завантаження	Дата початку: 25 липня 2021 року Дата завершення: 1 вересня 2021	Файл CSV повинен бути завантажений без лінії	Завантажено CSV-файл без рядків	Пройдено.
Перевірте експорт звітів у CSV	1. Відкрийте вкладку "Configurações" 2. Введіть дату початку експорту CSV 3. Введіть дату закінчення експорту CSV 4. Натисніть кнопку завантаження	Дата початку:1 січня 2021 Дата закінчення: 1 серпня 2021	Необхідно завантажити CSV-файл, який містить всі звіти за вказаний інтервал	CSV-файл, завантажений з усіма звітами з інтервалу	Пройдено.

Додаток Г

Результати тестових кейсів з резервного копіювання системи, відновлення версій резервних копій та керування конфігураціями системи.

Тестовий приклад	Етапи тестування	Дані тестув.	Очікувані результати	Фактичні результати	Пройдено/Не пройдено
Перевірте резервне копіювання системи, зокрема розкладів і пакунків	1. Відкрийте вкладку "Configurações" 2. Включіть пакунки та розклади 3. Натисніть на кнопку "Створити резервну копію	Ні.	ZIP-файл має бути збережений у директорії резервного копіювання, вказаній у налаштуваннях системи, після чого має з'явитися повідомлення про успіх	Zip-файл успішно збережено до каталогу резервної копії та успіх успіх повідомлення з'явилося	Пройдено.
Перевірте резервну копію системи, включаючи всі системні дані	1. Відкрийте вкладку "Configurações" " 2. Увімкніть всі опції 3. Натисніть на кнопку зробити резервну копію	Ні.	ZIP-файл має бути збережений у директорії резервного копіювання, вказаній у налаштуваннях системи, після чого має з'явитися повідомлення про успіх	Zip-файл успішно збережено до каталогу резервної копії та успіх успіх повідомлення з'явилося	Пройдено.
Перевірте резервну копію системи, з незаповненими полями конфігурації системи	1. Відкрийте вкладку "Configurações" " 2. Увімкніть всі опції 3. Натисніть на кнопку зробити резервну копію	Ні.	Повинно з'явитися повідомлення про помилку, яке інформує про необхідність заповнення всіх полів конфігурацій системи	З'явилося повідомлення про помилку, в якому зазначено, що всі поля повинні бути завершеним	Пройдено.
Перевірте відновлення резервної копії в системі, включаючи розкладом і пакетами	1. Відкрийте вкладку "Agendamentos" 2. Видаліть усі розклади 3. Відкрийте вкладку "Gestão de Pacotes" 4. Видаліть пакет з назвою "Pacote Exemplo" 5. Відкрийте вкладку "Configurações" 6. Виберіть резервну копію для відновлення з назвою, що закінчується підрядком "_sp" 7. Натисніть кнопку, щоб зробити відновлення	Ні.	Видалені розклади та пакунки мають бути знову в системі, і має з'явитися повідомлення про успіх	Графіки та пакети були відновлені, і з'явилося повідомлення про успіх	Пройдено.
Перевірте відновлення резервної копії в системі, включаючи всі дані	1. Відкрийте вкладку "Agendamentos" 2. Видаліть усі розклади 3. Відкрийте вкладку "Gestão de Pacotes" 4. Видаліть усі пакунки 5. Відкрийте вкладку "Configurações" 6. Виберіть резервну копію для відновлення з назвою, що закінчується підрядком "_csp" 7. Натисніть кнопку, щоб виконати відновлення	Ні.	Видалені елементи повинні знову з'явитися в системі, і повинно з'явитися повідомлення про успіх	Всі дані були відновлені, і з'явилося повідомлення про успіх	Пройдено.
Перевірте відновлення резервної копії в системі з незаповненими полями в конфігураціях системи	1. Відкрийте вкладку "Конфігурації"" 2. Виберіть резервну копію для відновлення 3. Натисніть кнопку, щоб виконати відновлення	Ні.	Повинно з'явитися повідомлення про помилку, яке інформує про необхідність заповнення всіх полів конфігурацій системи	З'явилося повідомлення про помилку, в якому зазначено, що всі поля повинні бути завершеним	Пройдено.
Перевірте оновлення конфігурацій системи	1. Відкрийте вкладку "Configurações" 2. Видаліть каталог резервної копії 3. Натисніть кнопку збереження	Ні.	Каталог резервних копій повинен бути порожнім	Каталог резервних копій порожній	Пройдено.

Додаток Д

Диск з роботою