Міністерство освіти і науки України
Харківський національний університет радіоелектроніки
Факультет комп'ютерних наук
(повна назва)
Кафедра програмної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Дослідження методів моделювання завадостійких кодеків для телекомунікацій
(тема)

Виконав:
здобувач 2 року навчання
групи ІПЗм-23-2
Віталій ПОЛИВ'ЯНИЙ
(власне ім'я, прізвище)

Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва спеціальності)

Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Керівник проф. Лариса ВЛАСЕНКО
(посада, власне ім'я, прізвище)

Допускається до захисту
Зав. кафедри _____ Кирило СМЕЛЯКОВ
(підпис) (власне ім'я, прізвище)

2025 р.

Харківський національний університет радіоелектроніки

Факультет комп’ютерних наук
Кафедра програмної інженерії
Рівень вищої освіти другий (магістерський)
Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва)
Тип програми освітньо-наукова програма
(освітньо-професійна або освітньо-наукова)
Освітня програма Інженерія програмного забезпечення
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«___» _____ 20___ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Полив'яному Віталію Івановичу
(прізвище, ім’я, по батькові)

1. Тема роботи «Дослідження методів моделювання завадостійких кодеків для телекомунікацій»

затверджена наказом університету від 15.04.2025р. № 290 Ст

2. Термін подання здобувачем роботи до екзаменаційної комісії 15.06.2025 р.

3. Вихідні дані до роботи: *загальнонаукові методи (спостереження, вимірювання та порівняння), методи аналізу та синтезу – для дослідження існуючих аналогів та визначення основних можливостей їх вдосконалення.*

4. Перелік питань, що потрібно опрацювати в роботі: *вступ, огляд та аналіз існуючих рішень, порівняльний аналіз аналогічних продуктів, постановка задачі, формування вимог до системи, вибір та обґрунтування елементів та технологій, структурна схема системи, реалізація бізнес-логіки, розробка інтерфейсу користувача, висновки, перелік джерел посилання, додатки.*

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Отримання завдання	16.04.2025	*виконано*
2	Аналіз предметної галузі і постановка задачі	17.04.2025 – 18.04.2025	*виконано*
3	Огляд та аналіз існуючих рішень	19.04.2025 – 22.04.2025	*виконано*
4	Формування вимог до системи	23.04.2025 – 25.04.2025	*виконано*
5	Структурна схема системи	26.04.2025 – 30.04.2025	*виконано*
6	Реалізація бізнес-логіки	01.05.2025 – 04.05.2025	*виконано*
7	Розробка інтерфейсу користувача	05.05.2025 – 09.05.2025	*виконано*
8	Аналіз отриманих результатів та формування висновків	10.05.2025 – 15.05.2025	*виконано*
9	Підготовка пояснювальної записки	16.05.2025 – 25.05.2025	*виконано*
10	Підготовка презентації та доповіді	26.05.2025 – 09.06.2025	*виконано*
11	Перевірка на плагіат	13.06.2025	*виконано*
12	Нормоконтроль	15.06.2025	*виконано*
13	Рецензування	16.06.2025	*виконано*
14	Попередній захист	17.06.2025	*виконано*
15	Занесення диплома в електронний архів	19.06.2025	*виконано*
16	Допуск до захисту у зав. кафедри	19.06.2025	*виконано*

Дата видачі завдання 16 квітня 2025р.

Здобувач _____ Віталій ПОЛИВ'ЯНИЙ
(підпис)

Керівник роботи _____ проф. Лариса ВЛАСЕНКО
(підпис) (посада, власне ім’я, прізвище)

РЕФЕРАТ / ABSTRACT

Пояснювальна записка до кваліфікаційної роботи: 101 сторінок, 16 рисунків, 6 таблиць, 30 джерел.

МОДЕЛЮВАННЯ КОДУ, ОБРОБКА СИГНАЛІВ, ЗБЕРІГАННЯ ДАНИХ КОРИСТУВАЧА, БЕЗПЕЧНИЙ ДОСТУП ДО СИСТЕМИ, НАЛАШТУВАННЯ ПРОЦЕСУ МОДЕЛЮВАННЯ.

Ця робота є важливою, оскільки існує лише обмежена кількість сучасних інструментів для вивчення моделювання завадостійкого кодування, яке все ще широко використовується в телекомунікаційних системах. Деякі з доступних рішень є універсальними і дорогими програмними пакетами, які зосереджені на складних обчисленнях і моделюванні. Інші рішення не відповідають вимогам якості і мають дуже обмежену функціональність.

Метою цієї роботи є розробка нових інструментів для покращення процесу дослідження завадостійкого кодування та забезпечення надійних функцій кодування для телекомунікацій. Предметом дослідження є вдосконалення процесу моделювання завадостійких кодеків та інструментів їх дослідження.

Для вирішення поставленої задачі були використані загальнонаукові методи, такі як спостереження, вимірювання та порівняння, а також методи аналізу та синтезу. Ці методи були використані для дослідження існуючих аналогів та визначення основних можливостей їх вдосконалення.

Результати роботи можуть бути використані для доповнення загальної розподіленої системи, що розробляється для освітніх цілей. Система може бути використана як модуль у більшій інтегрованій системі або як автономна операція. Дослідження та рішення, розроблені в даній роботі, можуть бути використані не тільки в науково-дослідницькому та навчальному процесах.

CODE MODELING, SIGNAL PROCESSING, USER DATA STORAGE, SECURE ACCESS TO SYSTEM, MODELING PROCESS CUSTOMIZATION.

This work is important because there are only a limited number of state-of-the-art tools for studying the modeling of interference-resistant coding, which is still widely used in telecommunication systems. Some of the available solutions are general-purpose and expensive software packages that focus on complex computation and modeling. Other solutions do not meet quality requirements and have very limited functionality.

The aim of this work is to develop new tools to improve the process of researching noise-resistant coding and provide reliable coding functions for telecommunications. The subject of the study is to improve the process of modeling noise-resistant codecs and tools for their research.

To solve this problem, we used general scientific methods such as observation, measurement and comparison, as well as methods of analysis and synthesis. These methods were used to study existing analogs and identify the main opportunities for their improvement.

The results of the work can be used to complement the overall distributed system being developed for educational purposes. The system can be used as a module in a larger integrated system or as a standalone operation. The research and solutions developed in this work can be used not only in research and education.

Завідувачу кафедри ПІ
проф. Кирилу СМЕЛЯКОВУ

ЗАЯВА

щодо самостійності виконання кваліфікаційної роботи та можливості її публікації (та/або публікації анотації кваліфікаційної роботи) в електронному архіві відкритого доступу ElAr KhNURE

Я, Полив'яний Віталій Іванович, здобувач вищої освіти на другому (магістерському) рівні вищої освіти академічної групи ІПЗм-23-2 кафедра програмної інженерії, заявляю: моя кваліфікаційна робота на тему «Дослідження методів моделювання завадостійких кодеків для телекомунікацій», що буде представлена в екзаменаційну комісію для публічного захисту, виконана самостійно, в ній не містяться елементи плагіату і вона може бути опублікована в репозиторії "ElArKhNURE". Погоджуюся з авторським договором, відповідно до Положення про репозиторій ХНУРЕ "ElArKhNURE". Всі запозичення з друкованих та електронних джерел мають відповідні посилання.

Я ознайомлений з вимогами академічної доброчесності, згідно з якими виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування дисциплінарних заходів.

10.06.2025 Віталій ПОЛИВ'ЯНИЙ

ЗМІСТ

ВСТУП

У наш час телекомунікації стали важливим засобом обміну даними. З початку 20-го століття відбулася швидка революція в електричних комунікаціях. З винаходом телефону, телеграфу, радіо і телебачення методи обміну інформацією на великі відстані стали більш поширеними. Сучасні інструменти дозволяють отримати доступ до інформації, що зберігається на серверах, розташованих за тисячі кілометрів, через Інтернет.

Однак з розвитком і популярністю засобів комунікації виникла проблема безпеки та надійності даних. Сьогодні Інтернет використовується для масового розповсюдження інформації. Для безпечного надсилання та отримання інформації використовуються різні методи, в тому числі шифрування вихідного сигналу. Шифрування - це перетворення інформації в упорядкований набір знаків, елементів і символів. Шифрування передбачає присвоєння заздалегідь визначеної комбінації кодів кожному повідомленню [1]. Однак жодна передача даних не застрахована від помилок, таких як радіоперешкоди, ненадійні канали зв'язку та збої в роботі обладнання. Для виявлення та виправлення помилок при передачі інформації використовується денойзинг.

Завадостійкі кодеки часто використовують для ненадійних каналів зв'язку, таких як телефонні лінії. При виявленні помилки такі канали просто не можуть повторно відправити повідомлення. Якщо в повідомленні є помилка, оригінальне повідомлення стає недоступним. Крім того, завадостійкі кодеки часто використовуються в системах зберігання інформації, включаючи оптичні та магнітні системи зберігання.

Тому завадостійке кодування є виправданим, але поточні методи дослідження не відповідають сучасним вимогам. Аналогові технології ще не досягли достатнього рівня технологічного розвитку, хоча є простір для вдосконалення та розширення. З метою детального вивчення процесу виявлення та виправлення помилок завадостійких кодеків було вирішено розробити систему, яка імітує їх роботу.

Результати роботи можуть бути використані для доповнення загальної розподіленої системи, що розробляється для освітніх цілей. Система може бути використана або як модуль у більшій інтегрованій системі, або як самостійна діяльність. Проведені дослідження та розроблені в роботі рішення можуть бути використані не тільки в науково-дослідницькому та навчальному процесах, але й у практичному застосуванні.

1 ОГЛЯД ТА АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ

Для того, щоб сформулювати вимоги до системи, що розробляється, необхідно проаналізувати вибірку подібних систем. Детальний аналіз допомагає створити список необхідних функцій та визначити сильні та слабкі сторони таких систем.

Наразі існує не так багато систем моделювання, які можна використовувати для дослідження функціональності завадостійких кодеків. Для перетворення циклічних кодів існує кілька універсальних і дорогих програмних пакетів розрахунку поля Галуа. Ці програми є універсальними і тому складними у використанні, оскільки вони мають справу з великою кількістю різних типів задач. Типовими прикладами таких пакетів є MatLab та Volfram Mathematica [2]. Тому ми розглядаємо використання простіших, подібних систем, які зосереджені на моделюванні.

1.1 Система моделювання кодеків XTest+

Мета системи – імітувати використання кодеків для освітніх цілей. XTest+ – це настільна програма, розроблена мовою програмування Java. Програма компілюється у виконуваний JAR-файл і може бути запущена на більшості операційних систем, якщо встановлено середовище виконання Java. Однак програма використовує велику кількість сторонніх бібліотек, тому її потрібно розповсюджувати як пакет. Вага не має значення, але для багатьох користувачів цей метод є непрактичним.

Система включає широкий спектр кодів, стійких до перешкод, таких як системні коди, циклічні коди та небінарні коди, а також кілька інших кодів. Основна мета системи – надати користувачеві теоретичну інформацію про існуючі коди, імітувати їх роботу та перевірити знання та практичні навички користувача.

Інтерфейс користувача настільної програми мінімалістичний та простий, але має деякі незначні недоліки. Після запуску програми користувачеві відображається головне меню зі списком доступних кодеків, розділених на відповідні категорії, а також чотири кнопки, які дозволяють переглядати

теоретичну інформацію, виконувати практичні завдання, запускати тести або генерувати звіти. Щоб скористатися функціями програми, користувач повинен вибрати потрібні кодеки зі списку, інакше кнопки для редагування кодеків будуть недоступні. Крім того, користувач не може змінити розмір вікна програми або збільшити розмір шрифту. Загалом, інтерфейс користувача не відповідає сучасним ергономічним вимогам [3]. Головне меню додатку зображено на рисунку 1.1.

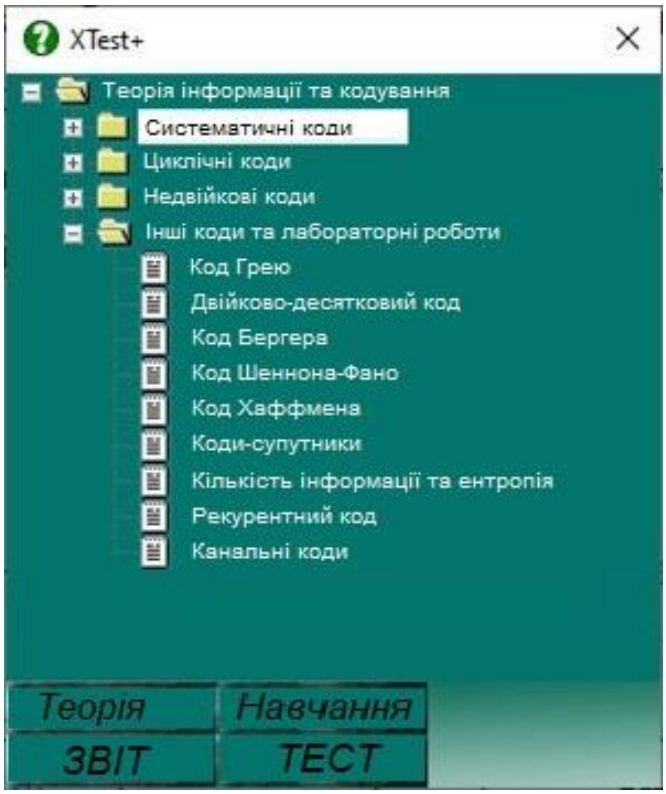

Рисунок 1.1 – Головне меню додатку XTest+ [3]

У розділі теорії користувач побачить теоретичну інформацію про вибраний кодек, необхідну для тестування. Більшість кодеків мають добре структуровану теорію та приклади кодування та декодування. Однак, у деяких кодеках відсутні приклади, а викладена теорія обмежується загальними описами, що ускладнює розуміння та засвоєння матеріалу.

Теоретичний матеріал відкривається в окремому вікні, яке, на відміну від головного меню, можна збільшити та залишити відкритим під час іспиту, що дуже зручно. Однак весь теоретичний матеріал представлений російською мовою, що може ускладнити розуміння теорії деяким користувачам. Наявність українського

перекладу або можливість зміни мови в додатку є корисною функцією для всіх користувачів, щоб забезпечити гарне засвоєння матеріалу. На рисунку 1.2 показано приклад вікна теоретичної інформації.

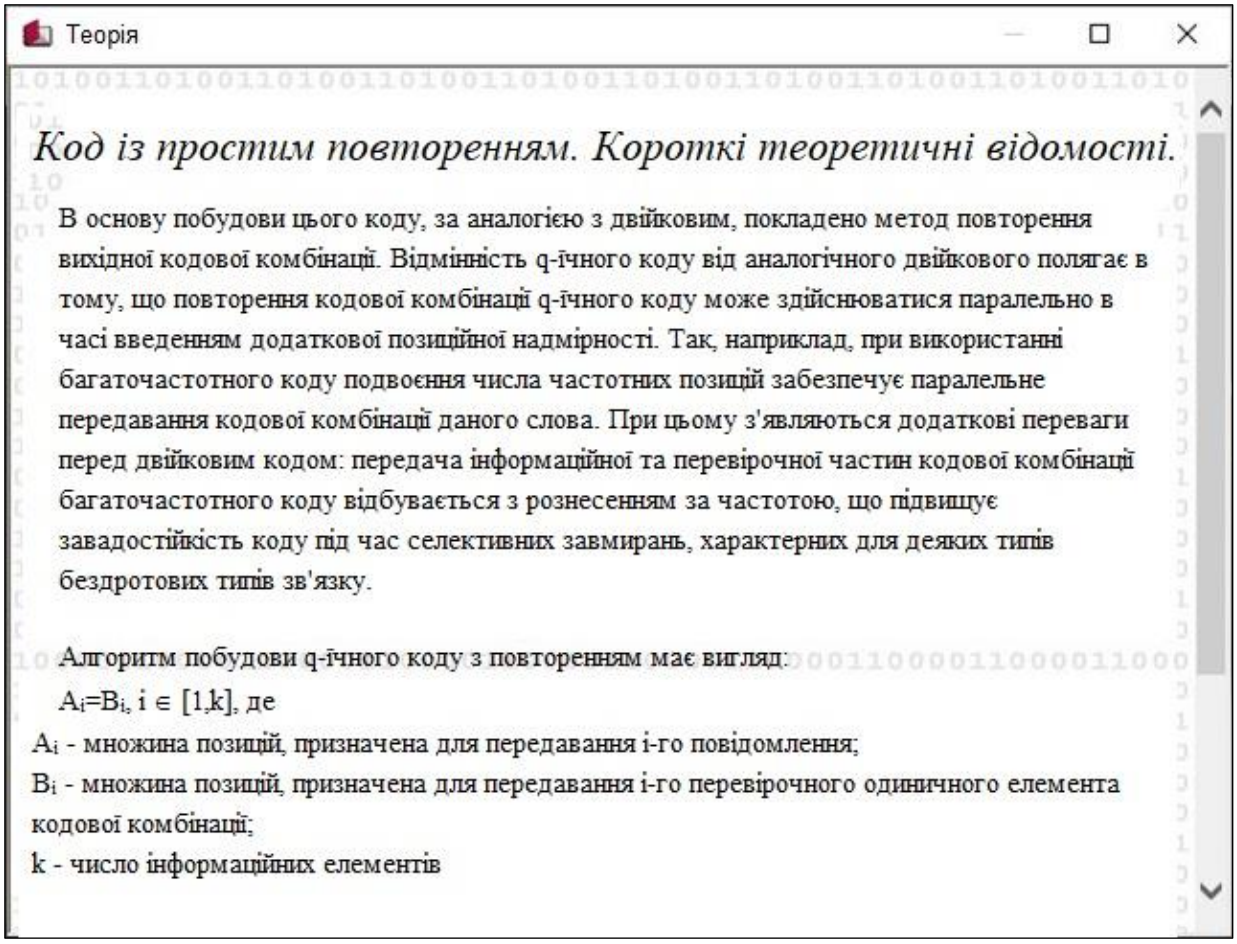

Рисунок 1.2 – Вікно з теоретичними відомостями коду [3]

Після ознайомлення з теоретичною інформацією про певний код, користувач може виконувати завдання кодування або декодування, щоб краще зрозуміти, як насправді працює кодек. Більшість реальних завдань представлені у вигляді невеликих вікон із випадково згенерованими умовами. Після введення відповіді користувач отримує повідомлення, яке підтверджує правильність виконання завдання.

Одним з основних недоліків системи XTest+ є неможливість зміни розміру шрифту. Зі збільшенням довжини умовного повідомлення стає все складніше підрахувати кількість символів, особливо якщо повідомлення у двійковому коді.

Крім того, генерація випадкових умов, безумовно, має переваги, оскільки жоден користувач не отримає одне й те саме завдання двічі. Однак для деяких кодеків, які містять кілька підтипів, користувач повинен закрити та знову відкрити вікно навчального завдання, оскільки функціональність інтерфейсу для вибору певного підтипу відсутня, доки не будуть виконані умови для потрібного підтипу коду. Корисною функцією систем моделювання є можливість змінювати певні параметри умов, за яких генеруються навчальні завдання. На рисунку 1.3 показано приклад навчального завдання.

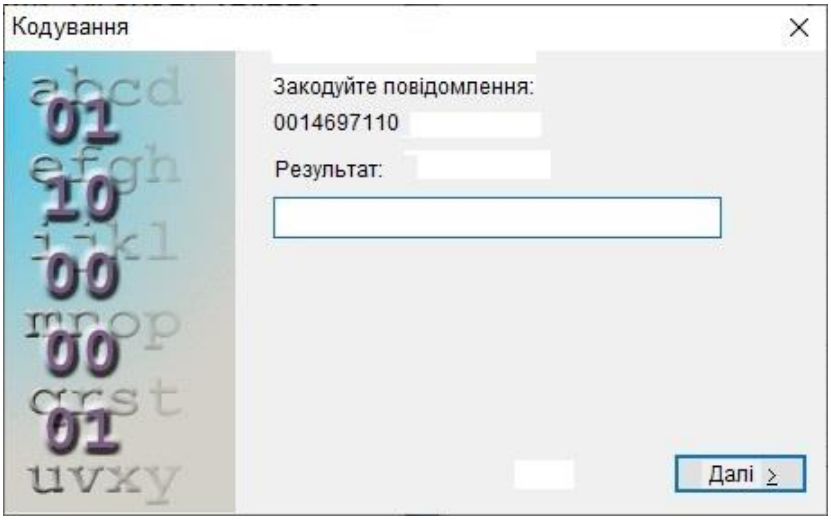

Рисунок 1.3 – Вікно з практичним завданням [3]

Після отримання практичних навичок користувачі мали пройти тест для підтвердження отриманих знань. Тест складався з десяти завдань: п'яти завдань кодування та п'яти завдань декодування, які були ідентичними навчальним завданням. На відміну від навчальних завдань, користувачам не повідомляється, чи виконали вони завдання правильно чи неправильно.

В кінці тесту користувачі отримують бал на основі кількості правильно виконаних завдань. Однак система не дозволяє переглядати неправильні відповіді. Результати тестів зберігаються під локальним іменем користувача на комп'ютері, на якому запущено настільну програму.

Користувачі можуть переглянути свої результати в розділі «Звіт», де зберігаються результати тестів. Система надає можливість друку цього звіту. В іншому випадку користувачеві слід зберегти скріншот звіту, оскільки програма

XTest+ не може зберігати дані користувача після цього сеансу. Вікно звіту показано на рисунку 1.4.

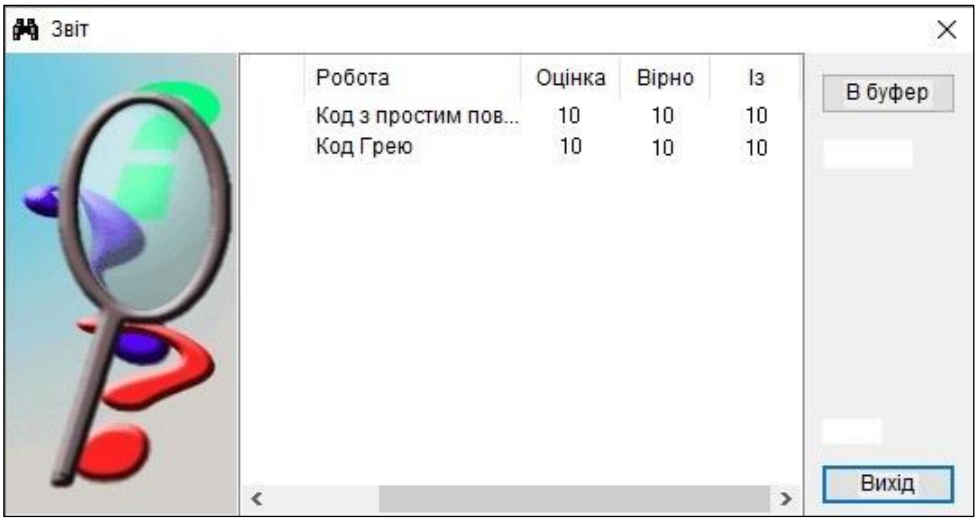

Рисунок 1.4 – Звіт виконаних тестувань [3]

Після закриття програми інформація про виконані вами тести видаляється і не може бути відновлена. Це серйозний недолік для настільної програми, призначеної для особистого використання. Система XTest+ не може підтвердити, що тест було фактично проведено, а також не може надати результат. Це робить настільні програми вразливими до програмного забезпечення, яке може маніпулювати пам'яттю вашого комп'ютера, такого як CheatEngine. Використовуючи такий пристрій, зловмисник може змінити звіти, що генеруються системою, та замінити їх бажаними значеннями.

Загалом, програмне забезпечення працює без затримок; нові вікна відкриваються швидко, а обчислення не займають надто багато часу. Існує невелика затримка після завершення тесту та збереження результатів, але це несуттєво.

Детальніше розглянемо десктопний застосунок, і можна побачити такі переваги системи XTest+:

- кросплатформність;
- легкий десктопний застосунок;
- широко використовувані кодеки;

– захоплення теоретичних даних та експериментів.

Недоліки включають:

– інтерфейс користувача застарілий та непрактичний;
– складний процес розповсюдження;
– дані користувача зберігаються лише протягом одного сеансу;
– обмежені можливості для виконання реальних завдань;
– умови генеруються випадковим чином;
– вразливість до програмного забезпечення, яке маніпулює пам'яттю комп'ютера;
– немає інформації про те, які тести були проведені;
– немає перекладу українською мовою.

1.2 Система моделювання і обробки сигналів SignalJ

Система SignalJ призначена для використання студентами для виконання практичних завдань з академічних предметів. Система надає широкий спектр можливостей для дослідження різноманітних сигналів, виконання математичних та інженерних операцій над ними, а також детального аналізу результатів. Основним завданням системи є створення та обробка сигналів за допомогою функцій, формул або вручну.

Система розроблена як настільний додаток з використанням мови програмування Java та скомпільована у виконуваний JAR-файл, що дозволяє програмі працювати на більшості операційних систем. Хоча SignalJ також спирається на зовнішні бібліотеки, на відміну від свого попередника, вони також компілюється в архіви .jar. Весь архів легкий, але дистрибутив програми не є практичним.

Інтерфейс досить складний, але інтуїтивно зрозумілий. Верхня панель містить основні функції програми – створення сигналів, редагування або обробка сигналів, відображення сигналів графічно або в таблиці, додаткові функції та налаштування. Під головним вікном розташована панель інструментів з

піктограмами для найпоширеніших дій – збереження, копіювання, вставка, закриття вікна тощо.

Для моделювання сигналів SignalJ надає широкі можливості для створення та налаштування джерел сигналів. Окрім визначених користувачем формул (які можна легко ввести в цьому інтерфейсі), ви також можете використовувати попередньо визначені стандартні типи сигналів без трудомісткого завдання їх опису за допомогою функцій. Система навіть дозволяє генерувати шум за допомогою різних параметрів. Меню генерації стандартних сигналів показано на рисунку 1.5.

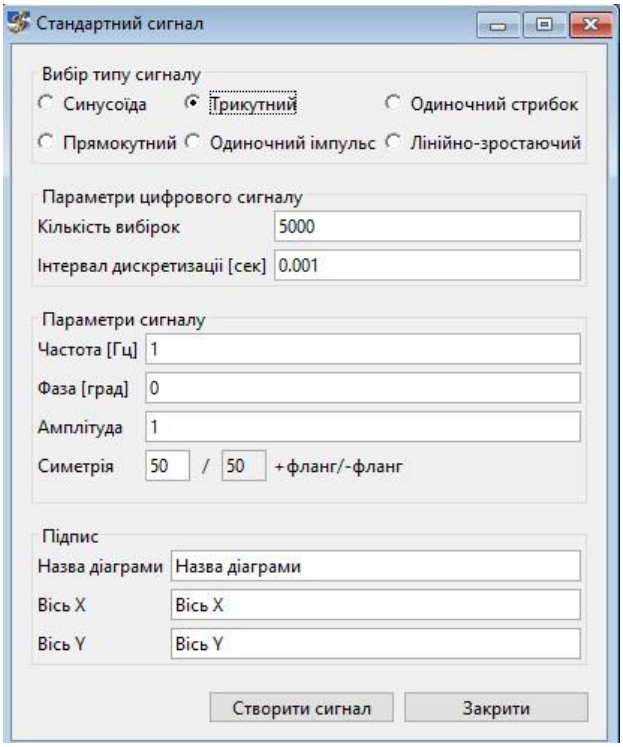

Рисунок 1.5 – Меню створення стандартного сигналу [4]

Після створення сигналу користувач може вибрати спосіб його відображення. Система дозволяє відображати сигнали у вигляді таблиці, лінійного графіка або гістограми. Однак деякі сигнали (наприклад, шум) неможливо відобразити певним чином або вони не мають значення у вибраному режимі відображення.

Користувач також може налаштувати відображення сигналу. Для графіків система пропонує можливість змінювати тип графіка та налаштовувати зовнішній

вигляд осей, стилів ліній, інтервалів та координатних сіток. Зручний інтерфейс дозволяє негайно застосовувати зміни та перемальовувати графік. Швидкість оновлення графіка залежить від складності формули, яка його визначає, та параметрів осей. Однак навіть у найскладніших випадках оновлення графіка займає лише кілька секунд, а продуктивність програми починає знижуватися, коли кількість відкритих вікон графіка перевищує 15. Сигнал відображається у вигляді лінійного графіка, а меню конфігурації показано на рисунку 1.6.

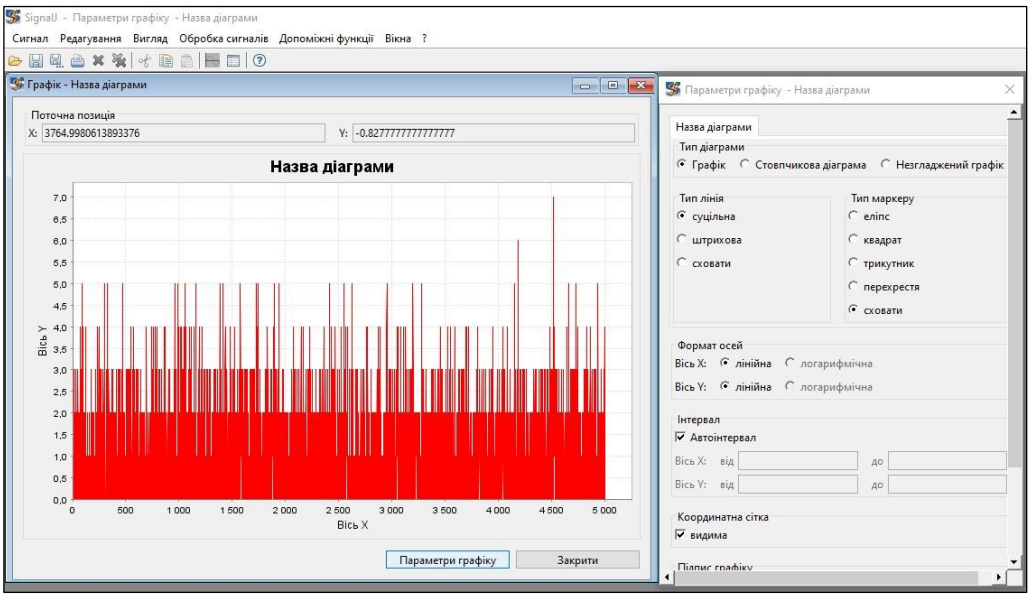

Рисунок 1.6 – Графік сигналу та меню його налаштування [4]

Генерація різних сигналів є лише частиною функціональності системи. SignalJ також дозволяє користувачеві виконувати різні дії зі згенерованими сигналами. Обробка сигналів розташована на вкладці «Обробка сигналів» і включає інтерполяцію, апроксимацію тощо. Після вибору та налаштування потрібної дії діаграма, що показує результати дії, відображається в окремому вікні програми.

Система надає повний інтерфейс налаштування для кожного типу обробки. Наприклад, для ряду Фур'є користувач може вибрати порядок гармонік, тривалість періоду та суму помилок, а також переглянути таблицю коефіцієнтів Фур'є та перерахувати ряд, якщо необхідно. На рисунку 1.7 показано графічне представлення апроксимації синусоїдального сигналу рядом Фур'є.

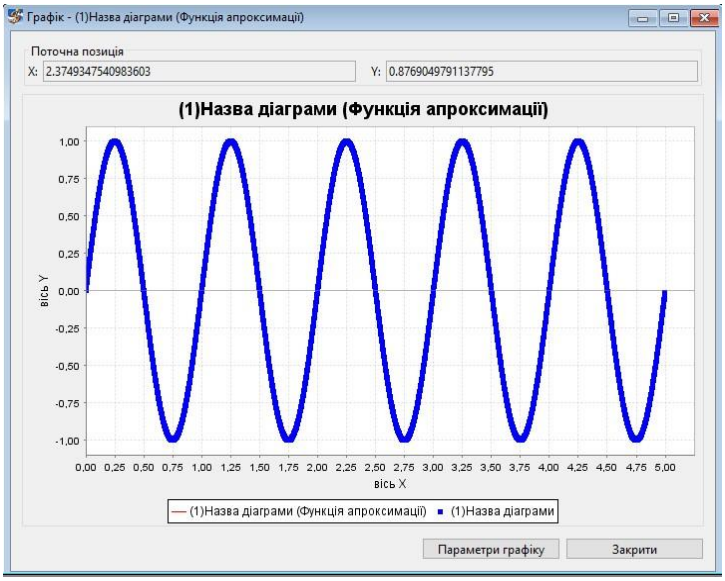

Рисунок 1.7 – Графік апроксимації за рядом Фур'є синусоїдного сигналу [4]

Однією з найбільших переваг системи моделювання SignalJ є можливість запису будь-якого сигналу. Дані, що використовуються для створення діаграми, зберігаються у файлах з розширенням .xml. Коли файл завантажується в програму, історія сигналів реконструюється на основі збережених даних.

Це дозволяє зберегти процес виконання завдання та відновити його пізніше, а також зменшує ризик втрати у разі випадкової помилки програми. Це дозволяє користувачам записувати свою роботу крок за кроком, щоб підтвердити, що завдання дійсно виконані. Однак більшість навчальних завдань вимагають роботи з великою кількістю сигналів та діаграм. Ручний запис та подальше завантаження їх у систему може бути нудним та громіздким. Немає функції автоматичного завантаження необхідних сигналів під час відкриття програми. Було б корисніше зберігати стан системи та всі налаштування сигналів і пов'язані з ними діаграми в окремому файлі.

Після детального опису системи SignalJ можна виділити такі переваги:

- комплексний функціонал системи;
- зручний та детальний інтерфейс користувача;
- можливість детального налаштування процесу моделювання сигналів;
- можливість збереження результатів роботи;
- висока швидкість розрахунків;

– можливість вибору методу представлення сигналу.

Недоліки:

– брак теоретичної інформації;
– непрактичність зберігання великої кількості сигналів та їх подальшого завантаження в програму;
– вразливість до програмного забезпечення, яке маніпулює пам'яттю комп'ютера;
– деякі елементи інтерфейсу користувача мають неточні трансляції.

1.3 Порівняльний аналіз аналогічних продуктів

Аналіз вищезгаданих аналогів виявив їхні позитивні та негативні характеристики. Ці фактори були використані для створення порівняльної таблиці між аналогами та системою, що розробляється (див. табл. 1.1).

Таблиця 1.1 – Таблиця порівняння аналогів (таблиця виконана самостійно)

Критерій оцінки	XTest+	SignalJ	Розроблювана система
Наявність теоретичних відомостей	+	–	+
Можливість налаштування процесу моделювання	–	+	+
Наявність тестування знань користувача	+	–	+
Збереження результатів роботи	+	+	+
Наявність українського перекладу	–	+	+
Зручний інтерфейс	–	+	+
Широкий функціонал системи	–	+	+
Гнучка архітектура для розширення системи	+	–	+
Кросс-платформеність системи	+	+	+
Можливість швидкого портування застосунку	–	–	+
Доступу до системи офлайн	+	+	–

Під час огляду та аналізу аналогів враховуються існуючі продукти, їх переваги та недоліки.

Хоча система моделювання коду XTest+ надає широкий спектр інструментів для моделювання коду та тестування, інтерфейс не є ні ергономічним, ні зручним для користувача, не зберігає дані користувача та не має багатьох важливих функцій, необхідних для такої системи.

Ми дійшли висновку, що система XTest+ більше не відповідає сучасним вимогам, а її розробка та підтримка є безглуздими.

Незважаючи на деякі недоліки порівняно з настільними програмами, система моделювання та обробки сигналів SignalJ надає практичний та детальний набір інструментів для обробки цифрових сигналів.

Переваги цієї системи переважують недоліки, і SignalJ можна вважати надійним прикладом системи, призначеної для моделювання перешкодостійких кодеків.

Обидві системи розроблені як портативні настільні програми, які не потребують встановлення на локальний комп'ютер користувача. Це скорочує час встановлення, але створює численні вразливості для програм, які звертаються до оперативної пам'яті комп'ютера.

Це підтвердило доцільність розробки нової, вдосконаленої системи.

Шляхом аналізу аналогів було визначено основні проблеми, які мала вирішити нова система: зберігання даних користувачів та результатів роботи за допомогою додатків та забезпечення їх безпеки та недоступності; забезпечення зручного інтерфейсу, що дозволяє користувачам змінювати процес моделювання кодеків; та гнучка архітектура з можливістю додавання нових модулів до готової системи.

Система також повинна бути платформонезалежною та мати український переклад, щоб нею могли користуватися всі користувачі. Ці критерії є надзвичайно важливими та визначають пріоритети під час процесу розробки. Інші менш важливі недоліки можна ігнорувати.

1.4 Постановка задачі

Метою даної роботи є розробка нових інструментів для покращення процесу аналізу роботи завадостійких кодеків. Для досягнення поставленої мети були визначені наступні завдання:

- вивчити основи завадостійкого кодування та декодування;
- проаналізувати існуючі аналогічні рішення та виявити особливості, що впливають на якість процесу моделювання;
- визначити необхідні вимоги до системи та, відповідно, основні підходи до розробки;
- оцінити найбільш підходящі архітектурні та технологічні рішення для системи та вибрати найбільш відповідне рішення для поставленої задачі;
- реалізувати кодування та декодування основних завадостійких кодів;
- використати отримані дані для розробки програмного забезпечення для системи моделювання завадостійкого кодування та декодування в телекомунікаціях, що покращить процес викладання та проведення досліджень;
- на основі отриманих даних розробити програмне забезпечення для системи моделювання завадостійкого кодування та декодування в телекомунікаціях, що дозволить покращити навчальний та науково-дослідний процес і зробити його більш практичним та доступним.

Предметом дослідження є робота завадостійких кодеків у телекомунікаціях. Об'єктом дослідження є процес моделювання та інструменти для дослідження вдосконалення завадостійких кодеків.

Для вирішення поставленої задачі були використані загальнонаукові методи, такі як спостереження, вимірювання та порівняння, а також методи аналізу та синтезу. Ці методи були використані для аналізу існуючих аналогій та визначення важливих методів покращення.

2 ФОРМУВАННЯ ВИМОГ ДО СИСТЕМИ

Формулювання вимог до системи, що розробляється, є важливим кроком у проектуванні програмного забезпечення. Вимоги визначають функції, характеристики та функціональність системи. Чітко сформульовані вимоги допомагають розділити процес розробки на логічні фази та визначити пріоритети впровадження функцій програмного забезпечення.

Вимоги поділяються на функціональні та нефункціональні вимоги. Спочатку ми визначимо функціональні вимоги, а потім нефункціональні.

2.1 Формування функціональних вимог

Функціональні вимоги описують поведінку та внутрішню функціональність програмного забезпечення. Як згадувалося в попередньому розділі, одним із питань, яке необхідно вирішити, є безпека користувачів, їхніх даних та роботи, що виконується в системі. Встановлюючи механізм авторизації через облікові записи, кожен користувач не лише отримує доступ до своїх даних, але й сприяє безпеці системи, що також є важливим питанням.

Крім того, система повинна пропонувати зручний інтерфейс та розширені можливості кодеків. Тому були визначені такі функціональні вимоги:

- забезпечити реєстрацію нових користувачів за їхньою електронною адресою;
- під час створення нового облікового запису зберігати особисту інформацію користувача;
- після створення нового облікового запису надсилати електронний лист із підтвердженням на електронну адресу користувача;
- забезпечити авторизацію існуючих користувачів;
- забезпечити функцію виходу з авторизованого облікового запису;
- після успішної авторизації надати користувачеві доступ до функцій системи;
- забезпечити наявність українських перекладів;

- надавати список кодеків, доступних для перегляду в системі;
- дозволити користувачеві вибрати потрібну іконку зі списку та перейти на сторінку;
- надавати можливість ознайомитися з відповідною теоретичною інформацією та прикладами роботи кодека;
- надавати можливість виконувати відповідні навчальні завдання;
- надавати можливість виконувати відповідні навчальні завдання в галузі кодування та декодування;
- надати можливість налаштування параметрів для створення умов виконання реальних завдань;
- надати можливість перегляду кількості виконаних навчальних завдань для вибраного коду;
- надати можливість проходити вибрані тести на кодування;
- надати можливість розрахувати час, необхідний для виконання тесту;
- надає можливість перегляду кількості виконаних навчальних завдань для вибраного коду;
- надає можливість скласти тест з вибраним кодом;
- обчислює час, необхідний для виконання тесту;
- оцінює тести, складені користувачами;
- надає можливість перегляду результатів тестування;
- дозволяє переглянути кількість спроб для вибраного тесту;
- надає можливість перескласти вибраний іспит;
- надає інформацію про цю версію системи;
- надає доступ до сторінок довідки.

2.2 Формування нефункціональних вимог

На відміну від функціональних вимог, нефункціональні вимоги визначають, як має працювати програмне забезпечення та які функції та можливості воно повинно мати [4, 5].

Для системи, що розробляється, нефункціональні вимоги описують показники якості програмного забезпечення. Для цієї системи були визначені такі вимоги:

- після завершення реєстрації протягом п'яти хвилин на електронну адресу користувача має бути надіслано повідомлення про успішне створення облікового запису;
- після успішної авторизації користувач має бути перенаправлений до головного меню системи протягом максимум однієї секунди;
- правильність адреси електронної пошти, наданої під час реєстрації, має бути перевірена;
- після виходу з облікового запису користувача система повинна повернутися до вікна авторизації протягом максимум однієї секунди;
- функціональність системи має бути доступна користувачам усіх операційних систем;
- повинно бути забезпечено, щоб час переходу з однієї сторінки системи на іншу становив не менше двох секунд.
- коректно завантажувати та відображати дані користувача щонайменше за дві секунди;
- створювати реалістичні умови місії визначеної користувачем тривалості, що не перевищує однієї секунди;
- похибка таймера, що використовується для вимірювання часу тестування, не повинна перевищувати трьох секунд;
- система повинна перевірити тест, виконаний користувачем, та надати відповідну оцінку протягом щонайменше двох секунд.

Таким чином, визначено необхідні системні вимоги шляхом попереднього аналізу сильних та слабких сторін подібних систем. Визначені вимоги поділяються на функціональні та нефункціональні.

Серед функціональних вимог виділяються ті, що зосереджені на вирішенні подібних системних проблем. Наприклад, авторизація користувачів у системі допомагає вирішити проблеми, пов'язані зі зберіганням та безпекою даних. Інші

вимоги стосуються базової функціональності системи. Система повинна забезпечувати можливість ознайомлення з теоретичним матеріалом, виконання практичних вправ, складання іспитів, отримання та зберігання оцінок. Крім того, система повинна не тільки зберігати дані користувачів, але й забезпечувати можливість перегляду цих даних. Іншими важливими вимогами були зручний інтерфейс та переклад українською мовою.

Щодо нефункціональних вимог, то основні вимоги стосувалися якості функціональності системи. Найважливішими параметрами оцінки розробленої системи є час виконання та точність даних. Система повинна швидко та точно обробляти запити. Надсилання повідомлень для інформування користувачів про авторизацію, умови місії, перевірку тестів, відображення даних користувачів та підключення до функцій системи має бути виконано протягом оптимального часу, щонайменше однієї секунди.

2.3 Сценарії використання системи

Варіанти використання є важливою частиною проектування системи. Вони визначають поведінку програмного забезпечення під час взаємодії з користувачем і спрямовані на досягнення конкретних цілей. Визначаючи системні прецеденти, можна створити чіткі алгоритми поведінки системи для різних дій користувача [6]. Таким чином, функціональність розробленої системи вже була детально розроблена на етапі проектування.

Визначте та опишіть варіанти використання на основі визначених функціональних вимог.

На основі вимог, визначених у попередньому розділі, визначаються основні сутності системи: - авторизовані користувачі; - неавторизовані користувачі.

Основні люди, які взаємодіють із системою, - це авторизовані користувачі. Після реєстрації нового облікового запису або входу в систему за допомогою існуючого облікового запису вони можуть отримати доступ до основних функцій системи. Після авторизації користувачі можуть переглядати список доступних

кодів, читати теоретичні розділи, виконувати практичні завдання, проходити тести, переглядати досягнення системи тощо.

Неавторизовані користувачі не можуть взаємодіяти з функціями системи, якщо вони не створять новий обліковий запис або не ввійдуть у систему за допомогою існуючого облікового запису. У таблицях Б.1-Б.12 додатка Б наведено детальний опис кожного прецеденту.

Таким чином, розглянуто визначення сценаріїв використання для системи, що розробляється. Як неавторизовані, так і авторизовані користувачі були визначені як основні зацікавлені сторони системи. Неавторизований користувач описує профіль гостьового користувача, який не має доступу до основних функцій системи та може лише зареєструватися або увійти. Після виконання цих кроків гість стає авторизованим користувачем з доступом до основних варіантів використання.

Основні варіанти використання включають реєстрацію, авторизацію, перегляд теоретичної частини, виконання практичних вправ, складання іспитів та перегляд власної роботи в системі: складені іспити, кількість спроб, бали та кількість правильно розв'язаних практичних вправ. Детальний опис кожного прецеденту визначає найважливіші, бажані та альтернативні способи його реалізації. Це забезпечує обробку помилок та необхідну функціональну гнучкість під час використання програмного забезпечення.

3 ВИБІР ТА ОБГРУНТУВАННЯ ЕЛЕМЕНТІВ ТА ТЕХНОЛОГІЙ

Перш ніж розпочати етап розробки програмного забезпечення, необхідно оцінити існуючі рішення, проаналізувати найбільш підходящі технології та вибрати ті, що забезпечать високу якість та швидке впровадження системи. Це включає вибір платформи розробки, типу програмного забезпечення, що буде впроваджуватися, мови програмування та основних фреймворків і бібліотек.

3.1 Обґрунтування архітектури системи

Як згадувалося вище, основні функціональні вимоги до системи можуть бути реалізовані в будь-якому представленні. Для цієї системи ми розглядали реалізацію у вигляді мобільного додатку, настільного додатку та веб-додатку. Давайте розглянемо кожен з них, щоб визначити, яка платформа підходить для системи, яку ви розробляєте.

Настільні додатки мають свої переваги, але вони значно переважують недоліки. Користувачі можуть встановити настільний додаток або завантажити архів та розпакувати його на своєму локальному комп'ютері. Система одразу готова до використання. Додаток простий у використанні, але, крім моментів, що враховуються під час аналізу аналогів, він має кілька недоліків, пов'язаних з його розробкою. Необхідність забезпечення незалежності додатків від платформи обмежує технології, які можна використовувати в їх розробці. Крім того, необхідно забезпечити безпеку даних, що є проблемою для суто офлайн-додатків[7]. Нечесні користувачі можуть використовувати спеціальні програми для фальсифікації інформації про результати тестів. Найкращий спосіб забезпечити цілісність даних – це використовувати окремий API для доступу до даних, що зберігаються поза програмою. Це, у свою чергу, призводить до низки проблем, пов’язаних із розподілом даних між системними модулями. Крім того, якщо система обмежена настільними програмами, додавання та розширення нових модулів може бути проблематичним.

Мобільні додатки мають схожі недоліки. Щоб забезпечити надійність даних, їх необхідно зберігати окремо від програми. Для забезпечення безпеки необхідно використовувати окремий API.

Крім того, вимоги кросплатформної сумісності обмежують використовувану технологію. Зрештою, хоча мобільні додатки є портативними та доступними з будь-якого місця, їхній обмежений розмір екрана перешкоджає використанню повної функціональності системи.

На відміну від розглянутих вище варіантів, веб-додатки можуть певною мірою вирішити вищезазначені проблеми. Доступ до функціональності веб-додатку здійснюється через браузер, що робить систему незалежною від платформи. Хоча в попередніх випадках доступ до Інтернету був обов'язковим, у цьому випадку він виправданий і усуває багато недоліків. Важливі дані захищені від атак і не можуть бути змінені шляхом доступу до пам'яті локального комп'ютера. Звичайно, дані, що надсилаються від клієнта до сервера, можуть бути перехоплені та пошкоджені. Однак вихідні дані зберігаються на сервері та можуть бути використані для відновлення [8].

Ще однією перевагою використання веб-додатку є те, що він має більш гнучку архітектуру, ніж настільний додаток. Таку систему можна розділити на окремі модулі, при цьому логіка виконання програмного забезпечення переміщується на сторону сервера, а рендеринг - на сторону клієнта. Такий підхід дозволяє розділити функціональність системи на мікросервіси, що робить архітектуру більш гнучкою. Система є супроводжуваною та розширюваною [9, 10].

Ще однією перевагою є те, що функціональність можна розділити на дві частини, серверну та клієнтську, що полегшує перенесення системи на інші платформи. Серверна частина містить бізнес-логіку системи, що дозволяє легко створювати різні клієнти для різних платформ. Оскільки цей підхід не прив'язаний до конкретного подання, система може мати кілька клієнтів, які використовують функціональність сервера. Наприклад, користувач з одним обліковим записом

може запускати тести в настільному додатку для Windows, переглядати результати в мобільному додатку для Android, а потім змінювати дані через веб-сайт.

Для реалізації клієнт обрав односторінковий додаток. Односторінковий додаток - це веб-додаток або веб-сайт, який взаємодіє з користувачами шляхом динамічного завантаження необхідної інформації на сторінку, а не завантаження всієї сторінки з сервера. Це дозволяє уникнути переривань при переході з однієї сторінки на іншу. Це робить додаток схожим на десктопний додаток. В односторінковому додатку весь необхідний код (наприклад, HTML, JavaScript і CSS) завантажується один раз разом з усією сторінкою, або необхідні ресурси завантажуються динамічно в міру необхідності.

Сторінки ніколи не перезавантажуються і управління ніколи не передається іншим сторінкам, але користувачі можуть вільно прокручувати інші сторінки, використовуючи історію, створюючи ілюзію, що додаток складається з багатьох частин. Це створює враження, що додаток складається з багатьох частин. Такий підхід знижує навантаження на сервер і оптимізує продуктивність системи. Клієнт звертається до сервера лише для оновлення певних елементів сторінки або для отримання, зберігання та модифікації необхідної йому інформації. Залежно від вимог системи, що розробляється, односторінковий додаток може бути найбільш підходящим рішенням [11].

3.2 Вибір веб-фреймворку

При реалізації веб-додатку особливо важливо вибрати відповідний веб-фреймворк. Сьогодні існують десятки інструментів для розробки веб-додатків. Кожен інструмент призначений для виконання певного завдання у веб-розробці, тому найбільш підходящий інструмент потрібно шукати з урахуванням вимог, згаданих вище. Ми відібрали найпопулярніші інструменти для аналізу та порівняння. jQuery - це швидка, невелика та універсальна бібліотека JavaScript. За допомогою цієї бібліотеки ви можете легко отримати доступ до будь-якого DOM-елемента, обробляти події, отримувати доступ і маніпулювати властивостями та

вмістом веб-елементів. Крім того, jQuery надає зручний API для використання Ajax [12].

Сильною стороною jQuery є те, що її легко використовувати під час розробки. Зазвичай бібліотеку можна використовувати, посилаючись на її файл розмітки. Якщо вам потрібна більша функціональність бібліотеки, ви можете вставити інші плагіни. Це дозволяє створювати функції JavaScript, необхідні для вашого проекту. Бібліотеку також можна інтегрувати за допомогою менеджера пакетів, наприклад, npm.

Недоліки jQuery полягають у наступному. Якщо ви пов'язуєте js-файл бібліотеки з кожною веб-сторінкою, яка її використовує, час рендерингу сторінки збільшиться, що призведе до зниження продуктивності всього веб-додатку. Цього можна уникнути, використовуючи CDN і кешування, але ці методи також мають свої недоліки. Плагіни jQuery UI досить обмежені у функціональності. Також відсутня внутрішня обробка залежностей і плагінів, наприклад, умовне завантаження виконання скриптів [13].

Отже, бібліотека jQuery, хоча й приваблива по-своєму, не підходить для розробки цієї системи. Оскільки функціональність системи реалізована на JavaScript, пов'язані з нею великі, неподільні JS-файли зменшують час виконання кінцевого продукту, що суперечить встановленим вимогам. Бібліотека jQuery підходить для простих анімацій, зміни стилів, Ajax-запитів та маніпулювання елементами сторінки. Однак, лише jQuery недостатньо для створення складних веб-додатків.

Згідно з опитуванням StackOverflow [14], веб-фреймворк Angular є одним з найпопулярніших інструментів серед розробників. Така висока оцінка зумовлена зростаючою популярністю односторінкових додатків, і фреймворк Angular, переписаний з AngularJS, спрямований на розробку таких додатків. Хоча Angular вважається фреймворком JavaScript, він рекомендує використовувати TypeScript, розширення синтаксису JavaScript, яке підтримує статичну типізацію.

Основною перевагою цього фреймворку є його компонентна архітектура. На відміну від AngularJS, який використовує архітектуру MVC, додатки Angular

поділяються на незалежні логічні та функціональні компоненти. Ці компоненти можна міняти місцями, комбінувати або використовувати в інших частинах програми. Така незалежність спрощує тестування та гарантує правильну роботу кожного компонента.

Angular та багато інших фреймворків використовують компілятори для перетворення вихідного коду та HTML у чистий JavaScript під час процесу збірки. Код компілюється до того, як браузер завантажить веб-застосунок, тому сторінки відображаються швидше. Ще однією перевагою Angular є його інтерфейс командного рядка. Він автоматизує процес ініціалізації, налаштування та конфігурації застосунку за допомогою простих команд.

Angular також надає інструменти для реалізації залежностей. Ці залежності визначають, як взаємодіють різні компоненти, і показують, як зміни в одній частині коду впливають на інші частини. Залежності роблять код читабельним та безпечним. Однак визначення залежностей між компонентами може зайняти багато часу.

Одним із недоліків цього фреймворку є те, що односторінкові додатки, які використовують JavaScript для зміни вмісту сторінки та тегів, мають погану SEO-оптимізацію. Крім того, Angular має досить велику інфраструктуру, яка включає багато різних функцій, які можуть бути не потрібні в простому проекті. Архітектура фреймворку складна, розгалужена та більше орієнтована на великі додатки. Крім того, ця архітектура обмежує гнучкість вибору інших технологій під час процесу розробки, що може бути суттєвим недоліком при використанні мікросервісного підходу [15].

Ще одним популярним інструментом для розробки веб-додатків є бібліотека React JavaScript. Вона зосереджена на створенні односторінкових додатків та мобільних додатків, оскільки найкраще підходить для отримання та обробки постійно змінюваних даних. Перевагою React є його відносно легкість. Складні веб-додатки можуть використовувати кілька бібліотек для керування станом додатка, маршрутизації та зв'язку зі сторонніми API.

Як і Angular, бібліотека React використовує простішу структуру компонентів. Компоненти React можна використовувати повторно та часто описують кнопки, списки та інший контент. Ці компоненти обгорнуті в компонент-обгортку тощо, аж до кореневого компонента. Така структура спрощує перегляд компонентів веб-сайту, а також розробку та тестування застосунку.

Ще однією перевагою React є використання віртуального DOM. Веб-застосунки зазвичай вимагають багато взаємодії з користувачем. Застосунки приймають дані, обробляють їх та відображають зміни. Оновлення DOM є ресурсомістким і, якщо його робити часто, може суттєво вплинути на продуктивність застосунку. Тому React використовує віртуальний DOM, який зберігає зміни, потім порівнює попередній та поточний стан елементів і знаходить найкращий спосіб їх зміни без оновлення всього DOM.

Однак ця бібліотека має такі недоліки. Користувацький інтерфейс компонента описано за допомогою JSX, декларативного XML-подібного синтаксису, що використовується з JavaScript. Цей підхід має свої переваги, але він обмежує обсяг коду, який можна написати для React. Шаблон не можна використовувати в інших програмах.

Крім того, незважаючи на широту та глибину екосистеми, деякі компоненти не отримують такого ж рівня підтримки, як основні бібліотеки, що може призвести до проблем розробки. Особливим недоліком є обмежений інтерфейс командного рядка, який не дозволяє глибоко налаштовувати проект [16].

Фреймворк Vue дуже популярний. Vue можна легко інтегрувати в проект за допомогою бібліотеки JavaScript, але його також можна використовувати як веб-фреймворк для розробки односторінкових програм. Vue включає багато найкращих функцій інших фреймворків, включаючи React. Vue має гнучку компонентну архітектуру та використовує віртуальний DOM. Як і React, Vue має багату екосистему, яка надає такі функції, як маршрутизація, керування станом тощо. Однак Vue офіційно підтримує компоненти своєї екосистеми та оновлює їх до останніх версій основної бібліотеки. На відміну від Angular, Vue можна

підключити до окремих частин існуючого проекту через CDN або мініфіковані JS-файли [17].

Компоненти, що використовуються Vue, включають HTML-шаблони, CSS-стилі та JavaScript-код. Це дозволяє імпортувати частини інтерфейсу користувача в іншу програму або використовувати препроцесор.

Однак Vue має деякі недоліки. Один з них – відсутність підтримки TypeScript. Враховуючи невеликий розмір Vue та загалом легку архітектуру, не завжди доцільно використовувати всі функції TypeScript. Однак підтримки недостатньо в ситуаціях, коли вона потрібна. Структури компонентів, що поєднують HTML, CSS та JS в одному файлі, можуть швидко стати нечитабельними [18]. Vue також бракує чітких архітектурних шаблонів. Особливо це стосується використання Vuex State Manager. З документації не зрозуміло, де саме має бути реалізована логіка додатку - в компоненті, окремому модулі чи менеджері станів.

Проаналізувавши доступні фреймворки, ми обрали Vue, який, на відміну від Angular, є легким, простим у використанні та має найкращі можливості бібліотеки React. Однією з найбільших переваг для системи, яку ми розробляємо, є чудова можливість розширювати додаток та додавати нові модулі. Використовуючи компонентну архітектуру Vue, нові модулі можна легко додавати до системи. Ми також вирішили використовувати компоненти з екосистеми Vue: Vuex для централізованого управління станом додатку, VueRouter для управління маршрутизацією, конфігурацією шляхів і навігацією, а також VueResource для обробки веб-запитів.

3.3 Вибір мови розробки

Рішення побудувати систему як веб-додаток за допомогою Vue змусило нас розглянути дві мови програмування - JavaScript і TypeScript. Vue дозволяє розробляти додатки будь-якою з цих мов. Тож давайте розглянемо ці дві мови ближче, щоб зрозуміти, яка з них більше підходить для реалізації системи.

Основною мовою програмування на стороні клієнта є JavaScript, яка разом з HTML і CSS є одним з найважливіших інструментів для побудови веб-сайтів і створення інтерактивних, динамічних сторінок. JavaScript підтримує прототипування, що відрізняє її від більшості традиційних мов, заснованих на класах. Його можна використовувати як мову програмування, так і об'єктно-орієнтовану мову. Це дозволяє розробникам використовувати JavaScript по-різному, залежно від поставленого завдання [19]. Гнучкість і функціональність JavaScript відповідає Vue.js, тому є сенс використовувати саме його.

Vue також забезпечує підтримку TypeScript. Однак інтеграція з TypeScript формально не підтримується, а синтаксис компонентів незграбний. На відміну від JavaScript, який інтерпретується під час виконання, TypeScript зменшує ймовірність помилок під час виконання, виявляючи помилки компіляції під час розробки, а також забезпечує сувору типізацію об'єктів для перевірки правильного типу змінних під час компіляції. Тож, якщо TypeScript – це краща версія JavaScript, чому б не використовувати його? Оскільки підтримка TypeScript у Vue погана і не покращиться до наступного релізу, не рекомендується використовувати його для розробки.

Крім того, Vue – це компонентна архітектура, де компоненти будуються як анонімні об'єктні літерали, тому більшість переваг TypeScript є надлишковими, а компіляція у звичайний JavaScript збільшить час виконання та знизить продуктивність системи. Беручи до уваги раніше визначені вимоги, JavaScript повністю відповідає основним вимогам і був обраний мовою програмування.

3.4 Інші технології

3.4.1 Firebase

Для зв'язку з бекенд-системою я вирішив використовувати VueResource, який є частиною екосистеми Vue. Оскільки більшість функціональності системи є клієнтською, такі інструменти, як Firebase, можуть бути використані для зменшення роботи на стороні сервера та задоволення певних умов.

Firebase надає кілька компонентів, які можна використовувати для заміни функціональності бекенд-системи. До них належать бази даних реального часу, системи авторизації, системи аналізу тощо. Перевагою Firebase є можливість його використання для веб- і мобільних додатків. Основною причиною використання цієї системи є відносно низьке навантаження на сторону сервера. Основними завданнями сервера в цій системі є авторизація користувачів і зберігання даних про тести та виконані завдання. Firebase вирішує цю проблему та надає кілька інструментів. Отже, немає потреби турбуватися про масштабування, продуктивність сервера та розмір бази даних; Firebase автоматично подбає про це. Ще однією перевагою цього сервісу є те, що його можна використовувати для розміщення веб-додатків [20].

3.4.2 Бібліотека компонентів

Існує багато популярних бібліотек, які надають попередньо створені елементи інтерфейсу для створення елегантних, приємних та інтуїтивно зрозумілих користувацьких інтерфейсів. Найпопулярнішою з них є Bootstrap. Bootstrap надає попередньо створені стилі CSS, які вбудовуються у ваш проект через CDN та можуть використовуватися для певних елементів. Бібліотеки, такі як Bootstrap, дозволяють швидко та ефективно розробляти користувацькі інтерфейси.

Оскільки ми використовуємо фреймворк Vue.js, ми вирішили використовувати бібліотеку, спеціально розроблену для нього. Найпопулярніші з них: - Vue Material; - Vue Material Components; - Vuetify. Усі вищезгадані бібліотеки можна легко інтегрувати в готовий проект через CDN, менеджер пакетів або як плагін за допомогою інтерфейсу командного рядка Vue [21]. Однак кожна бібліотека має свої обмеження.

Vue Material Components - використовує підхід Google Material Design та відносно прості компоненти. Ця бібліотека пропонує найменшу кількість попередньо створених елементів інтерфейсу на вибір. На відміну від інших варіантів, вона не інтегрується з Nuxt.js, що ускладнює візуалізацію на стороні сервера.

Vue Material має велику кількість попередньо створених компонентів. Ця бібліотека унікальна тим, що містить компоненти, розроблені спеціально для екосистеми Vue, такі як VueRouter, яка має кілька власних компонентів. Бібліотека також доступна в різних конфігураціях, що дозволяє ефективно імпортувати та використовувати цілі дизайни інтерфейсу користувача.

Vuetify є найповнішою та найзрілішою з вищезазначених бібліотек. Вона включає низку елементів інтерфейсу, від випадаючих меню до більш абстрактних елементів, таких як слайд-шоу. Кожен компонент Vuetify має гнучку логіку та багато опцій налаштування для адаптації використання компонента до конкретних обставин [22]. Бібліотека активно розробляється і, незважаючи на відсутність офіційної підтримки Vue, її функціональність покриває всі необхідні вимоги для розроблюваної системи. З цієї причини Vuetify була обрана як бібліотека компонентів.

3.4.3 Інструменти для роботи з модулями

Розробка веб-застосунків передбачає використання великої кількості сторонніх бібліотек, фреймворків та пакетів. Для належної комунікації потрібно використовувати відповідний менеджер пакетів. Цей інструмент відстежує версії пакетів, їх встановлення та використання. Для систем, що розробляються, важливо використовувати менеджер пакетів. Наразі найпопулярнішими є npm та yarn. Незважаючи на популярність та повсюдне поширення npm, yarn має файл блокування, швидше встановлення пакетів, і все це автоматично зберігається у файлі package.json. Однак NPM є найнадійнішим інструментом. Менеджер NPM має велику онлайн-базу даних публічних та платних приватних пакетів, яка називається реєстром NPM. Доступ до реєстру можна отримати через клієнт, а доступні пакети можна переглядати та шукати на веб-сайті npm. Офіційна документація Vue також рекомендує використовувати npm, тому ми використали цей менеджер пакетів.

Для компіляції пакетів та ресурсів, якими керує npm, вам потрібен конструктор модулів, який компілює файли пакетів у модулі JavaScript, які згодом

можна включити до веб-застосунку. Для цієї системи було обрано Webpack. Webpack є найпопулярнішим рішенням для цього завдання та може бути налаштований відповідно до ваших потреб. Однією з переваг цього конструктора є можливість динамічного імпорту, що добре працює з Vue [23]. Динамічний імпорт дозволяє розділити компоненти Vue на логічні модулі, які завантажуються під час першого використання. Таким чином, компоненти завантажуються за потреби, покращуючи продуктивність усієї програми.

3.4.4 Система контролю версій

Використання системи контролю версій є важливим у розробці програмного забезпечення. Можливість відстежувати покрокове впровадження окремих компонентів системи перетворює лінійний процес розробки на часову шкалу, де завжди можна повернутися на кілька кроків назад, щоб виправити помилки або скасувати зміни. Хоча вони не використовуються для командної розробки, вони є важливими інструментами для написання програмного забезпечення. Обрана система – Git. Ця система контролю версій є найбільш широко використовуваною та має зручний інтерфейс командного рядка, можливість безкоштовно розмістити власний веб-сайт та можливість створювати необмежену кількість приватних архівів.

Таким чином, проаналізовано існуючі технічні рішення для практичної реалізації системи. Було вирішено розробити систему як веб-застосунок. Це рішення пропонує значні переваги над іншими альтернативами та може бути поширене на інших клієнтів.

Для реалізації веб-застосунку ми проаналізували існуючі веб-фреймворки та бібліотеки, а також вибрали всі додаткові технології на їх основі. Для аналізу були обрані jQuery, Angular, React та Vue. Після аналізу стало зрозуміло, що Vue найкраще підходить для цієї системи завдяки своїй простоті використання, практичності, швидкій візуалізації компонентів та орієнтації на відносно невеликі проекти. Враховуючи потребу в архітектурі Vue, слабку підтримку TypeScript та відсутність сильних навичок письма, було вирішено використовувати JavaScript як

основну мову програмування. Щоб спростити створення користувацьких інтерфейсів, ми розглянули найпопулярніші бібліотеки компонентів Vue, такі як Vue Material, Vue Material Components та Vuetify. Останній пункт у цьому списку був обраний тому, що Vuetify має найбільший список попередньо створених компонентів та можливість їх детального налаштування.

Як згадувалося раніше, більша частина бізнес-логіки зосереджена на стороні клієнта, тоді як серверна сторона виконує операції авторизації, отримує та записує дані. Саме тому було прийнято рішення використовувати Firebase, який дозволяє делегувати серверні функції стороннім сервісам та зосередитися на самій програмі та її користувачах. Тому було обрано безсерверну архітектуру. Як інструменти для роботи з внутрішньою екосистемою ми обрали менеджер пакетів npm та збирач модулів Webpack. Обидва є найпоширенішими інструментами для своїх завдань та добре працюють з фреймворком Vue.js.

4 СТРУКТУРНА СХЕМА СИСТЕМИ

Розробка блок-схеми програми є одним із найважливіших кроків у процесі розробки програмного забезпечення. Блок-схема описує всі компоненти системи, їхні взаємозв'язки та те, як вони обмінюються інформацією чи іншими об'єктами. Добре розроблена блок-схема дозволяє легко модифікувати програмне забезпечення за потреби, якщо зміна покращить якість.

Під час створення структурної схеми було вирішено розділити систему на кілька модулів. Модулі можуть відрізнятися розміром, реалізацією та кількістю використовуваних компонентів, але в контексті блок-схеми було вирішено розглядати їх як частину однієї системи з певною функціональною логікою. Блок-схема розроблюваної системи показана на рисунку 4.1.

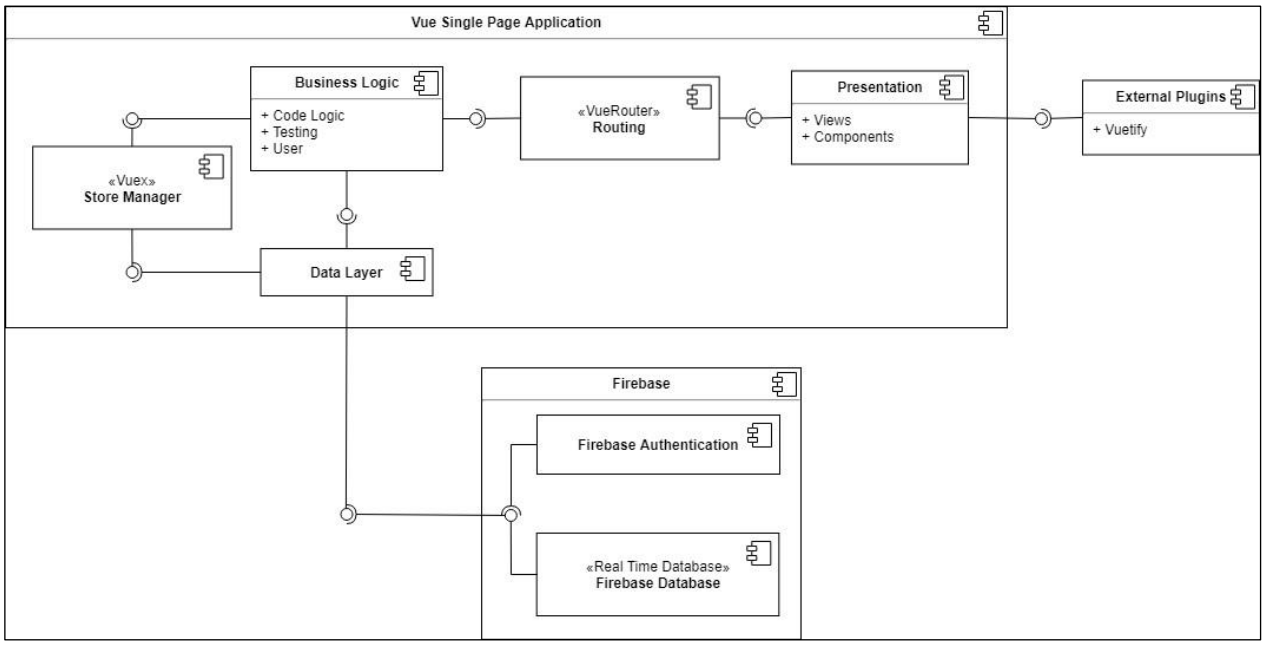

Рисунок 4.1 – Структурна схема системи (рисунок виконано самостійно)

Як показано на рисунку, систему можна розділити на дві частини: односторінковий додаток Vue.js та базу даних Firebase. База даних містить такі компоненти:

– автентифікація Firebase;

– база даних реального часу Firebase.

Односторінковий додаток складається з таких компонентів:

– рівень даних;
– менеджер магазину;
– сеанс кампанії;
– бізнес-логіка.

Крім того, система взаємодіє зі сторонніми демонстраційними бібліотеками, позначеними як зовнішні плагіни.

4.1 Компонент Firebase

На основі визначених вимог та сценаріїв використання було вирішено використовувати Firebase як рішення для зберігання даних. Ця база даних безпосередньо взаємодіє з односторінковим додатком, модуль автентифікації Firebase використовується для авторизації користувачів в системі, а база даних Firebase в режимі реального часу використовується для зберігання інформації. У базі даних зберігаються облікові записи користувачів, персональні дані, результати іспитів, виконані завдання тощо. Структура компонента Firebase показана на рисунку 4.2.

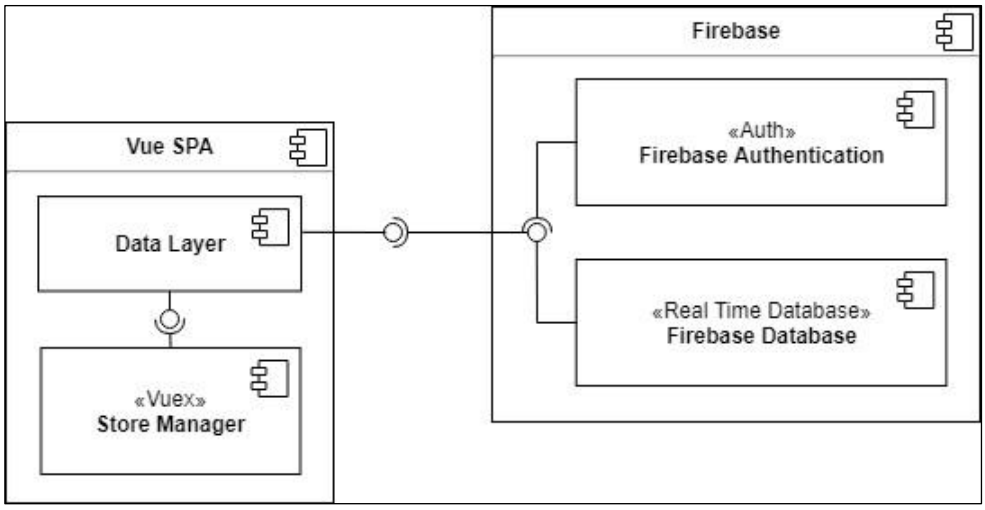

Рисунок 4.2 – Взаємодія компоненту Firebase з системою (рисунок виконано самостійно)

Взаємодія з базою даних відбувається через компоненти управління даними. Компонент рівня даних використовується для обробки загальних даних. Він взаємодіє з базою даних і отримує необхідну інформацію через протокол HTTPS.

Менеджер рівня зберігання відповідає за зберігання і розподіл даних, необхідних для роботи багатьох компонентів програми (наприклад, користувачів) під час сеансу. Більшість системних модулів працюють спільно з диспетчером сховища та рівнем даних.

4.2 Компонент Vue SPA

Односторінкові додатки можна розділити на три основні компоненти:

- компоненти бізнес-логіки;
- компоненти представлення;
- компоненти маршрутизації;
- компоненти управління даними та станом.

Структура компонентів показана на рисунку 4.3.

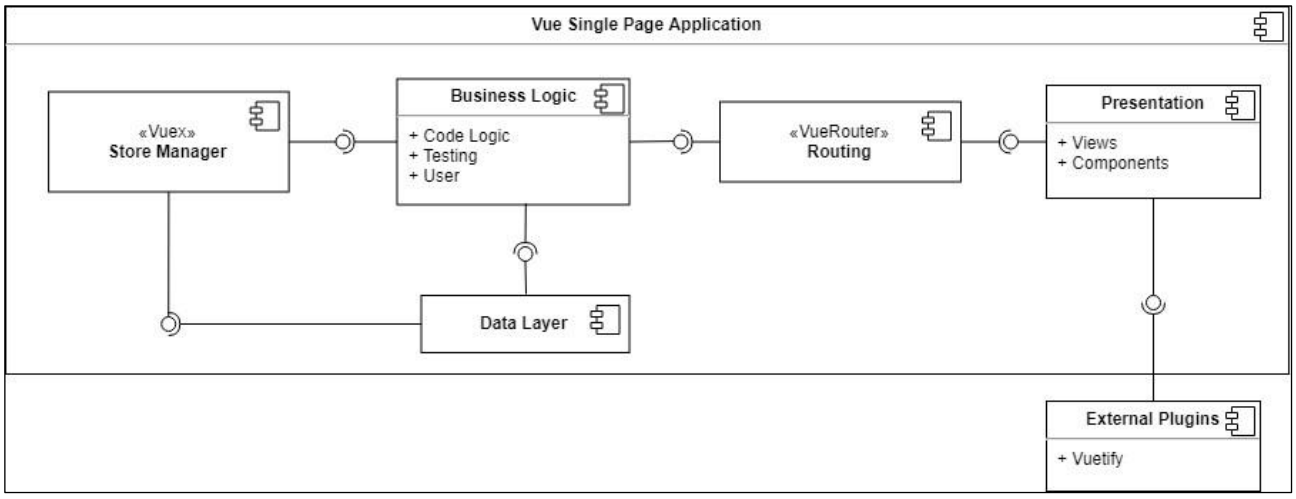

Рисунок 4.3 – Компонент Vue SPA (рисунок виконано самостійно)

Компоненти рівня даних та управління пам'яттю відповідають за взаємодію з джерелами даних та обмін інформацією з іншими частинами програми.

Модуль бізнес-логіки взаємодіє з цими компонентами. Цей модуль містить реалізацію всієї бізнес-логіки в додатку, а також відповідає за правильне представлення інформації на рівні даних.

Модуль бізнес-логіки складається з трьох окремих компонентів: Code Logic, Tests and Users.

Реалізація кодеків централізована в компоненті Code Logic, і кожен кодек представлений у вигляді окремого модуля. Це зберігає гнучкість системи, і нові кодеки можна впроваджувати без зупинки програми. Компонент Code Logic широко використовується іншими модулями системи.

Компонент тестування містить реалізацію генерації завдань, виконання тестів і перевірку функціональності завдань. Цей компонент взаємодіє з компонентами логіки коду та рівня даних. Завдання генеруються і тестуються за допомогою логіки коду, а результати тестування зберігаються в базі даних за допомогою рівня даних.

Користувацький компонент відповідає за логіку, яка обробляє дані, пов'язані з користувачем, отримує результати, авторизує тест тощо. Користувацький компонент взаємодіє з менеджером пам'яті, який зберігає дані користувача і заповнює їх у додатку, а дані користувача, що стосуються виконаних завдань і тестів, зберігаються в базі даних.

Модуль Route описує конфігурацію різних маршрутів додатку та пов'язаних з ними веб-сторінок. Це своєрідний модуль-з'єднувач між бізнес-логікою та компонентами презентації.

Для підготовки елементів інтерфейсу користувача ми використовуємо бібліотеку компонентів Vuetify, яка підключається до програми як зовнішній плагін, а плагін VueRouter використовується для маршрутизації програми. Рівень презентації отримує необхідну інформацію, таку як кількість завдань, виконаних користувачем, і представляє її у відповідному форматі.

Таким чином, представлена структура блок-схеми системи, яка розробляється. Визначено основні компоненти та модулі системи. Основним компонентом системи є односторінковий додаток, який містить більшу частину системної логіки. Він складається з компонентів даних і стану, компонентів бізнес-логіки, компонентів маршрутизації та компонентів презентації. Сторонні плагіни, такі як бібліотека Vuetify, також використовуються для розширення рівня презентації. Бізнес-логіка розділена на три логічні компоненти. Компонент «Логіка коду» містить кодування та декодування кожного коду візуалізації,

компонент «Тест» містить функціональність для генерації тестів і завдань, а компонент «Користувач» реалізує логіку для обробки даних, пов'язаних з користувачем, таких як отримані оцінки та виконані тести.

Firebase забезпечує функціональність сервера та бази даних. Цей модуль можна розділити на дві частини: авторизацію та базу даних. Firebase забезпечує та автоматизує процеси авторизації та зберігання даних. Односторінкові додатки взаємодіють з базою даних через HTTPS, використовуючи більш гнучкий та універсальний рівень даних, а також компонент керування сховищем, який відповідає за зберігання та розподіл даних, що використовуються в додатку.

Це забезпечує гнучку архітектуру, яка дозволяє системі масштабуватися за потреби.

4.3 Розроблення ER-діаграми

База даних Firebase реального часу була обрана як база даних на основі вимог. Дані Firebase зберігаються нереляційно в хмарі та оновлюються в режимі реального часу. Замість постійних HTTP-запитів використовується синхронізація даних. Це дозволяє всім клієнтам, які підключаються до бази даних, отримувати доступ до одного екземпляра сховища даних та автоматично отримувати оновлення, коли з'являється нова інформація. Це дуже корисно, коли багато користувачів обмінюються даними, оскільки всі можуть одразу побачити зміни. У цій системі ця функція використовується для розрахунку таймера тестування. Запис цієї інформації в режимі реального часу забезпечує її точність та повноту. Ще однією зручною особливістю бази даних Firebase є можливість доступу до інформації офлайн через локальне сховище даних, яке синхронізується після відновлення з’єднання [24].

Вся інформація в базі даних зберігається в нереляційному форматі JSON та у деревоподібній структурі. На відміну від баз даних SQL, Firebase не має таблиць чи записів. Коли ви додаєте дані до дерева JSON, вони стають вузлом у існуючій структурі з відповідним ідентифікатором. У такому вигляді їх легко використовувати на стороні клієнта, а запити можна обробляти легше та швидше.

Оскільки система не має складної структури, цей формат підходить. Враховуючи визначені для системи варіанти використання, було визначено такі сутності:

- User;
- CompeletedTest;
- ActiveTest;
- TestTask;
- PracticeRecord.

Сутність User описує обліковий запис користувача в системі. Вона містить загальну інформацію, таку як ім'я та група, інформацію, необхідну для авторизації системи, та статистику продуктивності користувача: виконані тести та вправи. Описи полів сутності наведено в таблиці 4.1.

Таблиця 4.1 – User (таблицю виконано самостійно)

Назва поля	Тип даних	Опис
id	String	Унікальний ідентифікатор
email	String	Електрона адреса користувача, використовується при авторизації.
name	String	Ім'я користувача, зберігається для загальної інформації.
group	String	Група користувача, зберігається для загальної інформації.
completedTests	Array	Виконані тести, зберігаються посилання на об'єкти.
completedPractice	Array	Записи виконаних тренувальних вправ.

Сутність CompletedTest описує завершений тест і використовується для запису статистики. Ця сутність містить результат і час, витрачений на завершення тесту. Вона також зберігає дату завершення завершених тестів і сортує їх за цією датою. Поля одиниць вимірювання описано в таблиці 4.2.

Таблиця 4.2 – CompletedTest (таблицю виконано самостійно)

Назва поля	Тип даних	Опис
id	String	Унікальний ідентифікатор
userId	String	Унікальний ідентифікатор користувача, який зберігається для зв'язку з об'єктом користувача
name	String	Назва кодеку з якого було пройдено тестування, зберігається для загальної інформації.
codecCode	String	Код відповідного кодеку, з якого було пройдене тестування.
result	Number	Результат пройденого тестування, зберігається для оцінювання.
timeUsed	String	Час, затрачений на виконання тестування, зберігається для загальної інформації.
Date	Date	Дата завершення тестування, зберігається для загальної інформації.

Елемент ActiveTest описує тест, який користувач наразі виконує. Він відрізняється від завершеного тесту тим, що записує згенеровані елементи та використовує таймер для підрахунку часу, необхідного для завершення тесту. Підтримка тесту активним у режимі реального часу дозволяє таймеру працювати безперервно та без помилок. Після завершення тесту активний тест зберігається в базі даних зі статусом «завершено». Поля сутності описано в таблиці 4.3.

Сутність TestTask описує фактичне завдання, створене для тестування. Ця сутність зберігає повідомлення, яке потрібно закодувати або декодувати, разом зі списком параметрів, необхідних для виконання. Поля сутності описано в таблиці 4.4.

Сутність PractiseRecord описує статистику виконаної вправи. Вона записує кількість задач, вирішених певним кодеком. Поля сутності описано в таблиці 4.5.

Таблиця 4.3 – ActiveTest (таблицю виконано самостійно)

Назва поля	Тип даних	Опис
Id	String	Унікальний ідентифікатор
userId	String	Унікальний ідентифікатор користувача, який зберігається для зв'язку з об'єктом користувача
Name	String	Назва кодеку з якого проходиться тестування, зберігається для загальної інформації.
codecCode	String	Код відповідного кодеку, з якого проходиться тестування.
timeLeft	Number	Час, який залишився для виконання тестування, зберігається в реальному часі.
timeTotal	Number	Загальний час, виділений на виконання тестування.
Tasks	Array	Згенеровані системою завдання для даного тестування.

Таблиця 4.4 – TestTask (таблицю виконано самостійно)

Назва поля	Тип даних	Опис
id	String	Унікальний ідентифікатор
testId	String	Унікальний ідентифікатор тестування, якому належить завдання, зберігається для зв'язку з батьківським об'єктом.
message	String	Згенерована умова завдання
params	Array	Додаткові параметри, необхідні для виконання завдання.

Таблиця 4.5 – PracticeRecord (таблицю виконано самостійно)

Назва поля	Тип даних	Опис
Id	String	Унікальний ідентифікатор
userId	String	Унікальний ідентифікатор користувача, який виконав дане завдання.
codecCode	String	Код відповідного кодеку, з якого була виконана тренувальна вправа.
amount	Number	Кількість виконаних користувачем тренувальних вправ.

Таким чином, описано базу даних Firebase Realtime Database, нереляційну хмарну базу даних, призначену для тестування. Вона забезпечує оновлення даних у режимі реального часу, синхронізує дані з підключеними клієнтами та використовується для керування таймерами під час тестування. База даних Firebase Realtime Database реплікує дані локально, що дозволяє системі працювати офлайн. Дані в базі даних зберігаються у форматі дерева JSON. Цей підхід найкраще підходить для простих сутностей веб-додатків – дані не потребують додаткової обробки на стороні клієнта, що ще більше пришвидшує запити до бази даних.

Сутності, що використовуються в системі, описуються як вузли в деревоподібній структурі Firebase. Найважливішими з них є Користувач, Завершений тест, Активний тест, Тестове завдання та Звіт про практику. Вони описують обліковий запис користувача в системі, завершені тести, тести, що виконуються, завдання та журнали вправ. Між усіма сутностями існують прості нерекурсивні зв'язки. Це робить базу даних масштабованою та легкою для розширення.

5 РЕАЛІЗАЦІЯ БІЗНЕС-ЛОГІКИ

5.1 Архітектура односторінкового додатку

Після аналізу вимог, вибору технологій та створення блок-схеми було вирішено використовувати безсерверну архітектуру та реалізувати клієнтський сайт як односторінковий додаток. Для подальшої реалізації була розроблена архітектура додатку.

Оскільки односторінковий додаток динамічно взаємодіє з користувачем, він завантажує необхідну інформацію з сервера на сторінку та завантажує всю сторінку вночі. Фреймворк Vue.js спрощує та робить розробку односторінкових додатків простішою та компактнішою. Однією з причин цього є розподілена, компонентна архітектура додатків Vue.js.

Кожен компонент позначено в окремому файлі .vue та містить шаблон HTML, що описує частину веб-сторінки, об'єкт Vue.js, що описує бізнес-логіку компонента, та стилі CSS, застосовані до шаблону.

Відправною точкою для односторінкового додатку є сторінка index.html, яка містить один блочний елемент із певним ідентифікатором. Основними функціями index.html є зберігання метаданих сторінки, підключення до сторонніх бібліотек та даних через CDN та визначення блочних елементів. Фреймворк Vue.js створює кореневий екземпляр Vue, який інтегрує всі плагіни та частини екосистеми Vue.js. Кореневий об'єкт – це окремий об'єкт Vue, створений за допомогою ключового слова new та описаний в окремому файлі main.js. Main.js налаштовує кореневий екземпляр та імпортує плагін.

Як кореневий компонент, фреймворк Vue.js використовує ідентифікатори елементів замість блокових елементів для відображення кореневого компонента App.vue. Ця сторінка відображає HTML-шаблон компонента. Кореневий компонент відрізняється від інших компонентів тим, що він містить усі інші компоненти та не може бути скопійований.

Архітектура компонентів дозволяє створити деревоподібну структуру для вашої програми, що дозволяє легко розширювати, додавати, видаляти та повторно використовувати модулі. Однак такий підхід має недоліки у міру зростання вашої

програми, а кількість модулів збільшується. Оскільки вся архітектура складається з компонентів, управління кожним компонентом стає все більш складним. Тому рамка VUE використовує метод розподілу компонентів у спеціальних пакетах [25].

Пакет містить усі плагіни, що використовуються в одній сторінці програми. Зазвичай всі плагіни імпортуються, налаштовані та використовуються у файлі main.js. Це зручно для роботи з невеликою кількістю плагінів, оскільки всі налаштування та імпортовані дані знаходяться в одному місці. Однак у міру збільшення кількості плагінів файли стають захараченими та важкими для читання. Оскільки я вирішив використовувати багато плагінів при розробці цієї системи, а саме Vuetify, Vuex, Vuerouter та Vueresource, було б зручно переміщувати файли конфігурації в окремий пакет. Коли ви додаєте новий плагін за допомогою Vue-Cli, файл конфігурації автоматично створюється та додається до пакету плагінів. Однак для великих модулів екосистем, таких як Vuex та Vuerouter, має більше сенсу створювати окремі пакети для магазинів та маршрутизаторів.

У програмі VUE на одній сторінці звичайна практика розділити всі компоненти презентації на дві категорії: компоненти, які відображають самі веб-сторінки, та компоненти, що описують елементи інтерфейсу цих сторінок. Ці компоненти зберігаються у двох різних пакетах - перегляди та компоненти. Оскільки компоненти презентації складаються з менших компонентів, це розділення полегшує визначення ієрархій компонентів.

Крім того, під час маршрутизації вашої програми конфігурація Vuerouter використовує компоненти, які описують цілі веб -сторінки. Таким чином, кожен шлях відповідає сторінці або компоненту.

Компоненти в пакеті компонентів - це, як правило, додаткові компоненти, які описують елементи інтерфейсу користувача, такі як панель навігації або випадаюче меню. Ця система використовує бібліотеку компонентів Vuetify, і більшість необхідних елементів інтерфейсу реалізовано. Однак для компонентів, які часто використовуються в додатку, було створено окремий пакет інтерфейсу користувача, який є підпакетом пакету компонентів.

Те саме стосується пакету макета, який містить компоненти, що описують різні частини веб-сайту, наприклад, HomeFooter.vue, HomeNav.vue. Таким чином, ці компоненти мають чітке призначення, їх легко розширювати і підтримувати в проекті.

Всі інші компоненти в пакеті компонентів належать до пакету підкодів. Вони описують інтерфейс і логіку, з якої складається сторінка в коді. Ці компоненти класифікуються відповідно до реалізованого коду і належать до відповідних пакетів коду. Різні компоненти реалізують теоретичну інформаційну сторінку та власне сторінку тестового кейсу. Використовуючи динамічний імпорт, ці компоненти автоматично імпортуються на активну сторінку і стають доступними для користувача. Це дозволяє розширювати систему новим кодом. Щоб додати новий модуль, необхідно додати реалізацію до відповідного пакета і реалізувати компоненти для кожного елемента коду.

Система автоматично перевіряє наявне сховище коду і додає новий код для використання. Більша частина бізнес-логіки є частиною пакета codeLogic. Він включає відповідні файли code.js, які містять логіку кодування та декодування. Ці файли можна імпортувати в компоненти для повторного використання логіки. Наприклад, компонент codec містить лише логіку представлення, тоді як функції кодування та декодування є окремими.

Пакет auth містить логіку авторизації користувачів у системі. Компоненти цього пакета перевіряють вхідні дані, надсилають запити на сервер і зберігають авторизованих користувачів сеансу в менеджері стану додатку. Фреймворк Vue.js не має власного шаблону представлення з власною логікою, особливо коли він взаємодіє безпосередньо із зовнішніми сервісами, такими як авторизація, незалежно від компонентів. З цієї причини функція авторизації винесена в окремий файл, який є частиною окремого пакету. Структура модулів системи показана на рисунку 5.1.

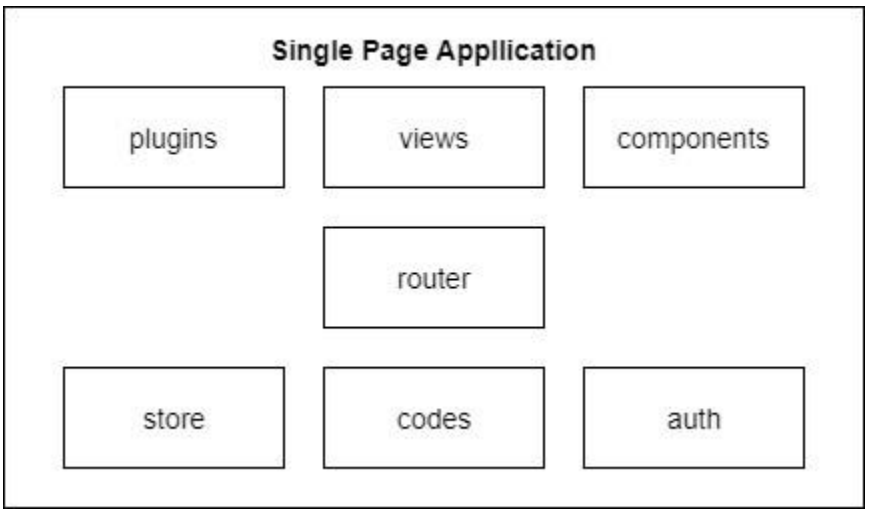

Рисунок 5.1 – Структура модулів пакетів застосунку (рисунок виконано самостійно)

5.2 Використані шаблони проектування

5.2.1 Шаблон проектування MVVM

Ефективна і проста розробка розподілених і масштабованих систем вимагає використання патернів проектування. Патерни надають готові рішення проблем, які виникають при написанні програмного забезпечення і є необхідними для розробки великомасштабних продуктів. Для даної системи були обрані наступні патерни проектування.

Патерн MVVM був розроблений як варіація патерну Presentation Model і фокусується на відокремленні розробки користувацького інтерфейсу від бізнес-логіки. Він створює чітку систему поділу, в якій обов'язки можуть бути розподілені між різними частинами програми [26].

Патерн проектування MVVM складається з трьох частин:

- подання, яке є користувацьким інтерфейсом додатку. У даній системі це подання представлено компонентами, що належать до пакету View and Components;
- модель, яка є власне моделлю, що містить дані, необхідні для функціонування додатку. У цій системі модель представлена користувачем, кодом і тестовими даними, що зберігаються в базі даних;

– ViewModel, яка є абстракцією представлення і пакетом даних для взаємодії між моделлю і представленням. ViewModel також відстежує зміни, які користувач вносить до даних, і розробляє логіку роботи подання.

Шаблон MVVM підходить для додатків, які передбачають прив'язку даних, і оскільки було вирішено використовувати фреймворк Vue.js, його застосування для цієї системи цілком виправдане.

Веб-фреймворк Vue.js зосереджується на частині шаблону ViewModel та зберігає її суть. Vue.js пов'язує модель з представленням через двонаправлений обмін даними. Кожен об'єкт Vue є ViewModel, частиною, яка з'єднує дві інші частини шаблону.

Для окремих компонентів ViewModel – це сама DOM або елементи, якими керує екземпляр Vue. Кожен об'єкт пов'язаний з відповідним елементом на веб-сторінці, тому Vue.js використовує шаблони на основі DOM. Коли створюється екземпляр Vue, ініціалізується програма або створюється компонент, він рекурсивно перебирає всі дочірні елементи елемента, з яким він пов'язаний, та встановлює необхідні зв'язки даних. Після компіляції представлення воно реагує на зміни в моделі.

Це усуває необхідність безпосереднього маніпулювання елементами DOM. Коли дані змінюються, представлення автоматично оновлюється за допомогою об'єктів Vue. Елементи представлення оновлюються дуже точно, аж до окремих вузлів. Це покращує продуктивність односторінкових програм, оскільки змінюються лише окремі частини, а не вся сторінка.

Модель шаблону Vue.js – це модифікований анонімний об'єкт JavaScript або об'єкт даних. Ці об'єкти належать компонентам Vue та описуються як функції, що повертають об'єкти даних. Це гарантує, що кожна модель компонентів є унікальною, тобто всі компоненти посилаються на один і той самий об'єкт даних.

Коли ви створюєте об'єкт Vue і відповідний йому об'єкт даних, екземпляр створює відповідні властивості для кожного поля об'єкта, щоб відстежувати зміни і повідомляти екземпляр Vue, коли потрібні оновлення. Це робить модель

реактивною. З Vue немає необхідності вручну перевіряти зміни та оновлювати подання. На рисунку 5.2 більш детально показано, що представляють собою шаблони для кожного компонента [27].

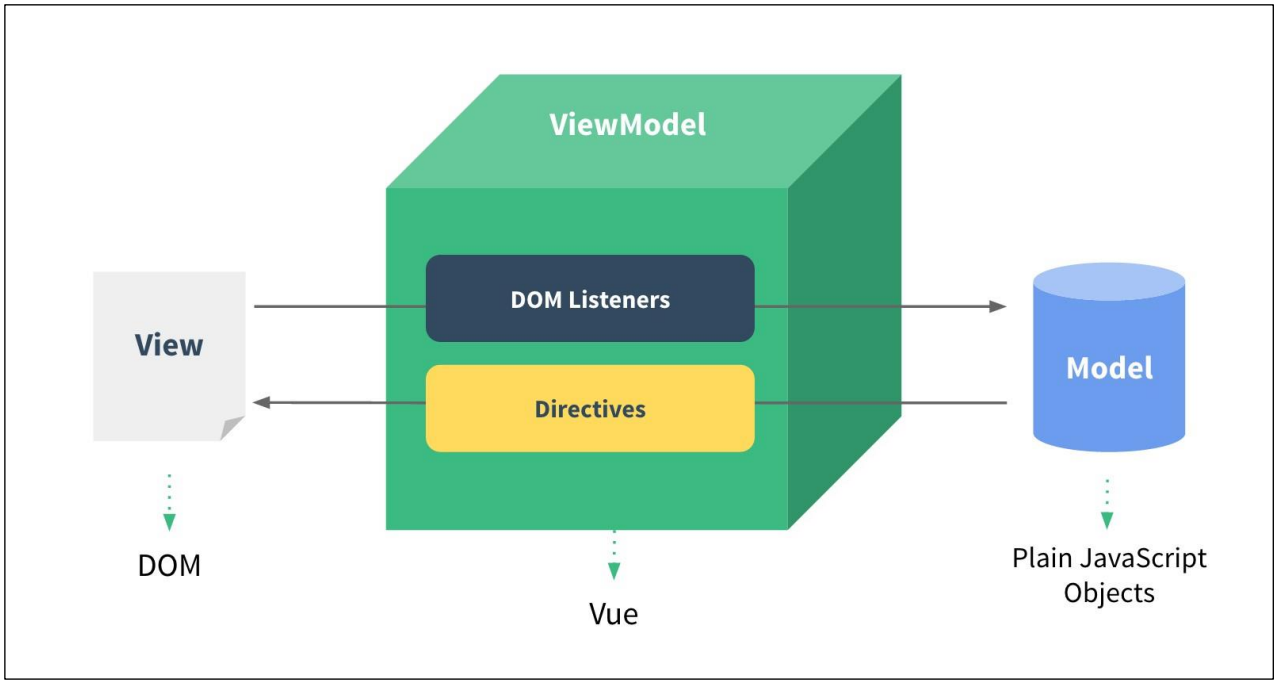

Рисунок 5.2 – Шаблон MVVM у Vue.js (рисунок виконано самостійно)

Отже, кожен компонент Vue реалізує патерн проектування MVVM, використовуючи двостороннє зв'язування даних і слухачів подій. Це створює систему, в якій подання оновлюється, коли змінюється модель, а модель реагує на зміни у поданні, а Vue виступає в ролі посередника. Оскільки Vue.js використовує компонентну архітектуру, кожен елемент вкладки стає частиною шару ViewModel.

5.2.2 Шаблони реалізації односторінкових додатків Vue.js

Для реалізації цієї системи ми використовували необхідні патерни реалізації, рекомендовані для додатків на Vue.js [28]. Ці методи розробки спрямовані на підвищення якості написаного коду та спрощення роботи багатьох модулів системи. Найважливішими з них є:

- використання специфічних компонентів;
- модульний підхід;
- стан додатку;

– стилі компонентів.

Оскільки вся архітектура односторінкового додатку Vue.js базується на компонентах, для розробки великої та складної системи необхідно реалізовувати компоненти зі специфічними вимогами. Загальна методологія розробки диктує, що більшість компонентів повинні бути багаторазовими, здатними виконувати конкретні завдання і не залежати від архітектурного рішення.

Це означає, що компоненти повинні бути розроблені для конкретної мети і не повинні мати внутрішнього доступу до стану додатку або сторонніх ресурсів. Якщо додати таку логіку в компонент, він, по-перше, стає громіздким, а по-друге, втрачається весь сенс повторно використовуваних компонентів.

Саме тому ми застосували модульний підхід до цієї системи. Замість того, щоб концентрувати бізнес-логіку в компонентах, що відповідають за логіку користувацького інтерфейсу, ми винесли більшу частину функціоналу в окремі файли та модулі. Наприклад, вся бізнес-логіка для кодування та декодування, авторизації, генерації тестових завдань, активності користувачів та функціональності для роботи зі штатними адміністраторами зберігається в окремих JS-файлах. Це дозволяє розширювати, повторно використовувати та додавати нові функції програми.

Великі програми, де обмін даними є складнішим, ніж односторонній обмін між батьківськими та дочірніми компонентами, потребують спільного сховища даних, доступного для всіх компонентів програми. Vuex State Manager надає таке сховище даних. Файли для налаштування та використання функціональності State Manager включені до пакету репозиторію. Використання Vuex може призвести до переміщення частин бізнес-логіки програми з компонентів до State Manager. Щоб запобігти цьому, система, що розробляється, визначає набір правил для обробки стану програми.

По-перше, всі дані, що зберігаються у Vuex, можуть зберігатися лише в одному місці. Такі компоненти, як UserPage.vue, Code.vue тощо, використовують лише ці дані та не створюють власних копій для керування ними. Це зберігає цілісність даних. По-друге, Vuex зберігає лише ті дані, які потрібно розподілити

по всій програмі. У даній системі це дані про авторизованих користувачів. Компоненти не можуть самостійно змінювати стан програми; лише авторизовані пакети мають доступ до цього стану. Таким чином, використання Vuex не означає передачу всієї бізнес-логіки публічному адміністратору, але дозволяє уникнути складності обміну інформацією між компонентами.

Розробка клієнтської частини веб-додатку та збільшення кількості і розміру сторінок дозволяє створювати велику кількість файлів CSS. Управління та підтримка цих файлів може зробити розробку більш складною. У цій системі ми використовуємо бібліотеку компонентів Vuetify, тому кількість класів CSS невелика. Коли ці класи потрібно використовувати, ми описуємо їх в окремому компоненті і використовуємо атрибут scope, щоб обмежити їх використання. Таким чином ми уникаємо нагромадження класів CSS. Глобальні класи використовують окремий глобальний файл CSS.

5.3 Авторизація користувача з використанням Firebase та Vuex

Автентифікація користувачів реалізована в системі за допомогою Firebase Authentication, а для розподілу даних використовується Vuex State Manager. Логіка авторизації реалізована в державному менеджері та в модулі автентифікації.

Реєстрація користувачів реалізована в компоненті RegisterForm.vue реєстраційної форми. Для створення облікового запису в системі користувач повинен вказати адресу електронної пошти, ПІБ, номер групи, пароль та поля для перевірки пароля. Перевірка даних застосовується до полів адреси електронної пошти, пароля та підтвердження пароля. Компонент полів Vuetify має вбудований механізм перевірки, який використовує правила, описані в окремому файлі formRules.js. Витягнувши їх в окремий файл, їх можна застосувати до будь-якої форми в додатку без необхідності переписувати правила. Ці правила перевіряють правильність введеної адреси електронної пошти, відсутність порожніх полів, правильність довжини пароля та перевірку поля пароля. Вони також описують повідомлення, яке отримає користувач, якщо певне правило буде порушено. Якщо введена інформація не відповідає цим правилам, користувач отримає

повідомлення, а форма буде позначена як недійсна. Firebase надає необхідні інструменти для реалізації авторизації різними способами. У цій системі ми використовуємо авторизацію за допомогою електронної пошти та пароля, оскільки Firebase автоматично пропонує можливість надсилання повідомлень для підтвердження реєстрації, зміни пароля тощо.

Для коректної роботи програми необхідно використовувати дані авторизованого користувача з сеансу в різних компонентах. Щоб уникнути складного обміну даними між компонентами, ми вирішили перенести частину логіки авторизації в Vuex Status Manager. Функціональність менеджера описана в модулі зберігання і поділяється на чотири основні функції:

- статус;
- геттери;
- мутації;
- дії.

Статус зберігає поточний стан програми, яким у цій системі є змінна authUser, що описує авторизованого користувача. Геттери описують функціонал для доступу до статусу, мутації дозволяють змінювати стан програми і зазвичай недоступні поза контекстом Vuex, а дії описують логіку адміністратора.

Дія реалізує методи signUpUser, signInUser та signOutUser, які описують реєстрацію, авторизацію та вихід користувача. Компоненти RegisterForm та LoginForm викликають ці методи через this.$store. Під час реєстрації методи Firebase використовуються для створення користувача з вказаною адресою електронної пошти та паролем. Користувач зберігається в Firebase, йому присвоюється унікальний ідентифікатор, а його статус записується за допомогою методу setUser. Для автентифікації користувача метод Firebase отримує обліковий запис з бази даних, використовуючи адресу електронної пошти та пароль, вказані користувачем у компоненті LoginForm.vue. Облікові дані користувача зберігаються у локальному сховищі localStorage, щоб користувачеві не потрібно було щоразу вводити їх повторно. Логіка роботи цього сховища реалізована в модулі auth.

Інші компоненти можуть отримати доступ до зареєстрованих даних користувача через функцію getter менеджера статусів. appHeader.vue використовується для відображення цієї інформації на панелі навігації. userPage.vue використовується для відображення даних користувача та іншої інформації.

5.4 Реалізація логіки кодеків

Як згадувалося раніше, функції кодека були передані на аутсорсинг окремому модулю codeLogic. Це відокремлює основну частину коду від логіки компонента, яка описує конкретний тест або завдання кодека. Окремі файли, що містять необхідні фрагменти коду, пов'язані та використовуються в компоненті за потреби.

Модуль CodeLogic поділено на чотири підмодулі: systematicCodes, cyclicCodes, nonbinaryCodes та otherCodes, які охоплюють відповідно систематичні, циклічні та небінарні кодеки. Такий поділ дозволяє логічно структурувати існуючі файли відповідно до типу шумового кодека. Кожен підмодуль містить js-файли, які описують реалізацію кодування та декодування кожного кодека.

Кожен js-файл фактично описує компонент Vue без шаблонів HTML. Цей компонент може містити всі властивості екземпляра Vue: об'єкти даних, методи, спостерігачі та обчислювані властивості. Додаючи такий файл до компонента як атрибут власного міксину Vue, компонент може використовувати всі властивості та є вільним від екземпляра Vue.

Кожен міксін модуля CodeLogic містить два основні методи: Code та Decoder, які реалізують логіку кодування та декодування коду відповідно. Ці методи зазвичай приймають повідомлення про те, що компонент був відформатований та перевірений міксером, а також додаткову інформацію, необхідну для належного функціонування кодека, таку як розмір алфавіту для переважно недвійкових кодів або спряжена матриця для простих ітераційних кодів. Крім того, кожен кодек має метод checkAnswer для перевірки відповіді

користувача щодо кодування та декодування. Цей метод використовує методи кодування та декодування та повертає логічне значення, що вказує на помилку або правильну відповідь. Деякі коди мають додаткові методи для виклику допоміжних методів. Деякі коди вимагають обчислень або додаткових інструментів кодування. Таким чином, кожен файл має унікальну та специфічну структуру, яка демонструє функціональний підхід Vue.js.

5.5 Генерація практичних завдань та оцінка тестування

Компоненти, що використовують файли модулів codeLogic, поділяються на дві групи: Навчальні компоненти та компоненти для тестування користувачів, такі як GreyCode.vue, GreyDecode.vue та GreyTest.vue для Grey Code. Вправи на кодування та декодування є окремими.

Компоненти кодування та декодування описують інтерфейси для конкретних навчальних завдань. Зазвичай вони містять поле зображення з інформацією, яку потрібно згенерувати, і, за необхідності, додаткові інформаційні поля. Перед відправленням відповіді користувач має можливість вибрати, яку інформацію повинна перевірити система. Якщо введена користувачем інформація відповідає вимогам цього коду, завдання розпізнається. Система також може генерувати випадкову інформацію на основі вибраних користувачем параметрів. Для цього можна використовувати різні елементи інтерфейсу користувача, наприклад, повзунки, списки, що випадають, тощо. Таким чином, ви можете налаштувати завдання інструменту так, щоб користувачеві було легше розібратися з кодеком.

Після того, як користувач підтвердив відповідь, компонент порівнює її з відповідним файлом коду. Щоб переконатися, що виконані тестові завдання є значущими для користувача, система фіксує кількість правильно виконаних завдань для кожного кодека. Якщо відповідь правильна, система повідомляє про це користувача та оновлює його обліковий запис даних. Транзакції Firebase захищені спеціальними правилами, які налаштовуються через консоль Firebase. Ці правила забезпечують безпеку з'єднання і запобігають доступу до бази даних ззовні програми.

Це дозволяє користувачам бачити прогрес розробки під час виконання реальних завдань. Якщо відповідь неправильна, користувач може продовжити виконання завдання або скасувати процес розв'язання. Якщо користувач скасовує процес, він отримує нове завдання і всі параметри конфігурації скидаються до значень за замовчуванням, в іншому випадку відповіді та умови зберігаються, і користувач може продовжувати роботу.

Функціональність компонента Test аналогічна функціональності самого компонента Task. Для опису активного тесту створюється об'єкт ActiveTest. Кожен тест містить п'ять елементів кодування та п'ять елементів декодування. Логіка генерації тестових умов міститься у файлі TestGenerate.js, який використовує модуль codeLogic для отримання параметрів, необхідних для створення умов. Завдання, згенеровані цим компонентом, нічим не відрізняються від тих, що виконуються користувачем у самому компоненті. Однак, на відміну від них, користувач не може змінювати умови та параметри, а також не отримує сповіщення про правильність виконання завдання після його завершення. Кожен фрагмент коду має власний таймер, який відраховує час тестування. Коли тест запускається, таймер запускає зворотний відлік до його завершення. Значення таймера зберігається в компоненті. Однак, щоб уникнути ситуації, коли користувач закриває та знову відкриває вкладку браузера, таймер тестування зберігається в базі даних. Оскільки Firebase дозволяє зберігати дані в режимі реального часу, значення таймера в додатку синхронізується з даними в базі даних.

Якщо користувач не завершить тест вчасно, він отримає оцінку за виконане завдання або буде змушений почати все спочатку. База даних записує кількість спроб, необхідних для виконання тесту для кожного фрагмента коду. Кількість спроб необмежена, але зберігається для цілей звітності.

Після завершення тесту система збирає відповіді в масив і перевіряє їх по черзі. Потім система створює об'єкт CompletedTest з об'єкта ActiveTest, який зберігає результати тесту, дату проведення тесту і час, що минув. Це дозволяє користувачеві в будь-який момент перевірити хід виконання тесту і, при бажанні,

підтвердити, що тест пройдено. У компоненті UserResults.vue є архів, який містить історію виконаних тестів і вправ.

Таким чином, розглянуто реалізацію бізнес-логіки системи. Оскільки система була реалізована як безсерверний односторінковий додаток на Vue.js з компонентною архітектурою, ми вирішили розділити компоненти додатку на модулі. Були створені наступні модулі: auth, codeLogic, store, router, components, view.

Для взаємодії між шарами представлення та даних компонентів Vue використовується патерн проектування MVVM. У цій системі представлення та компоненти, що описують елементи інтерфейсу користувача, реалізовані у вигляді представлень. Дані користувачів, облікові записи користувачів, тестові дані та виконані вправи, що зберігаються в базі даних, використовуються як моделі. Коннектор - це екземпляр Vue, який з'єднує моделі та подання за допомогою подій та двостороннього обміну даними.

Авторизація користувачів здійснюється за допомогою служби Firebase, Vuex State Manager і модуля Auth. Користувачі можуть зареєструватися і увійти в систему за допомогою своєї електронної адреси та пароля. Дані користувачів зберігаються в Vuex State Manager, до якого має доступ весь додаток, щоб забезпечити узгоджений обмін даними між компонентами.

Модуль CodeLogic містить логіку виконання для кодування і декодування. Кожен код реалізований в окремому файлі, який потім зв'язується з необхідними компонентами, функції яких використовуються для управління фактичним завданням або тестом. Функція генерації завдань реалізована в окремому файлі TestGenerate.vue, який використовується для тестування компонентів. Після проходження тесту запускається таймер, значення якого зберігається в Firebase для забезпечення безперервної роботи.

6 РОЗРОБКА ІНТЕРФЕЙСУ КОРИСТУВАЧА

Розробка інтерфейсу користувача є важливою частиною процесу впровадження програмного забезпечення. Зручний дизайн спрощує взаємодію користувачів із функціями системи. Оскільки система розроблена для навчання симуляціям, простий, зрозумілий, ефективний та зручний інтерфейс користувача забезпечує кращий досвід використання програмного забезпечення.

6.1 Реалізація компонентів програми

Раніше було вирішено використовувати бібліотеку Vuetify для стандартних компонентів відповідно до специфікації Material Design. Використання Vuetify для розробки інтерфейсу користувача спрощує розробку на стороні клієнта та дозволяє нам зосередитися на логіці представлення програми. Оскільки Vuetify використовує стандартні шрифти та значки, ми включили необхідні файли через CDN. Vuetify також надає велику кількість користувацьких класів та класів на основі bootstrap для глибшого налаштування компонентів.

Компоненти, що використовуються для розробки інтерфейсу користувача, можна класифікувати наступним чином:

- компоненти розмітки та контейнери;
- компоненти для розробки інтерфейсу користувача;
- компоненти для групування елементів;
- компоненти для форм, збору та введення даних;
- компоненти для навігації;
- компоненти для кнопок та індикаторів;
- спливаючі та модальні вікна.

Компоненти розмітки визначають положення інших елементів інтерфейсу на сторінці. Вони використовуються для розділення сторінки на секції. Кожен компонент Vue може містити лише один кореневий елемент, і ці компоненти діють як обгортки. v-container - це загальний контейнер, який не має багато властивостей, що налаштовуються. У цій системі він використовується для

централізованого групування цілих сторінок і створення записів. v-row, v-col і v-flex є більш загальними і специфічними у своєму використанні, і в основному використовуються для групування невеликих елементів інтерфейсу.

Компоненти, що групуються, утворюють вужчі групи, ніж контейнери. Вони також дозволяють впорядковувати елементи на сторінці, але, на відміну від контейнерів, мають деяку вбудовану функціональність. Найпоширенішим компонентом є vcard, яка фактично описує готову сторінку. Vcard є багатофункціональним компонентом, і при розробці цього інтерфейсу він використовувався для написання форм авторизації та реєстрації, а також кодових компонентів. Іншими компонентами є v-tabs та v-list, які використовуються для створення динамічних тегів для власне завдань кодування та декодування, а також для написання списків реалізованих кодеків.

Компоненти форм переважно використовуються для відображення полів вводу та виводу. Якщо ви хочете реалізувати власну форму, ви можете скористатися попередньо визначеними компонентами v-textfield і v-form. Ці компоненти розширюють подібні елементи HTML і надають зручні функції для виконання користувацьких операцій. Наприклад, форми мають вбудовану валідацію, якою можна керувати безпосередньо на стороні клієнта. Компоненти v-форм дозволяють користувачеві здійснювати навігацію в додатку.

Компоненти навігації дозволяють здійснювати навігацію всередині програми. Компоненти v-меню та v-панелі інструментів є контейнерами для посилань на інші сторінки системи, які можна легко згрупувати.

Кнопкові та екранні компоненти описують інтерактивні елементи інтерфейсу, з якими взаємодіє користувач. До них відносяться різні типи кнопок, панелей інструментів тощо.

Діалогові компоненти використовуються для надання користувачеві необхідної інформації та опцій, компонент V-Dialog описує діалогове вікно, яке можна використовувати як обгортку для будь-якого іншого компонента, а компонент CodeCompleteDialogue.vue використовує V-Dialog для створення модального вікна, де користувач може вибрати, продовжувати чи ні після введення

відповіді на вправу. Використовуючи стандартний інструмент Vuetify, ми реалізували власні компоненти, з яких складаються веб-сторінки додатку.

Всі реалізовані компоненти вигляду поділяються на пакети вигляду та компоненти, тобто компоненти, що описують веб-сторінки, та компоненти, що описують окремі елементи користувацького інтерфейсу:

- HomePage.vue;
- LoginPage.vue;
- RegisterPage.vue;
- UserPage.vue;
- CodePage.vue;
- InfoPage.vue.

Компоненти веб-сторінок Компонент HomePage.vue описує головну сторінку додатку. Він складається з компонентів AppHeader.vue, Home.vue та CodeList.vue. Головне меню додатку показано на рисунку 6.1.

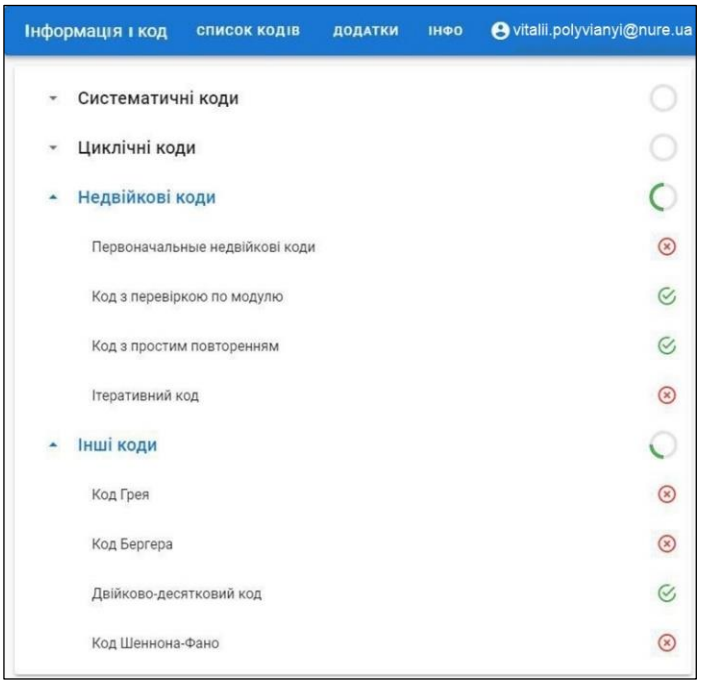

Рисунок 6.1 – Головне меню додатку (рисунок виконано самостійно)

На головній сторінці є панель навігації, за допомогою якої можна перейти на сторінку довідки, сторінку користувача, список кодів або повернутися до головного меню. Ви також можете використовувати панель навігації, щоб вийти з

системи і повернутися на сторінку входу. На головній сторінці відображається список доступних кодів, які користувач може розшифрувати, а також відображається прогрес користувача в системі.

Компонент LoginPage.vue описує форму входу для існуючого облікового запису користувача. Вона містить поля для введення адреси електронної пошти та пароля, а також кнопку підтвердження. Поле пароля має спеціальну маску, створену Vuetify для приховування введених даних.

Компонент Register.vue описує форму реєстрації нового облікового запису. Ми використовуємо компоненти v-row і v-card, а також класи bootstrap для створення необхідного макету сторінки. Компонент v-form використовується для групування та валідації реєстраційних полів. Компонент v-text-field використовується для налаштування зовнішнього вигляду повідомлення та перевірки його за допомогою атрибуту rule. У полях пароля і перевірки пароля можна використовувати спеціальні маски для приховування символів, що вводяться. Крім того, компонент v-text field має додаткове поле, яке відображає повідомлення, якщо під час перевірки виникла помилка. Кнопка «Відправити» неактивна доти, доки форма не буде валідована.

Компонент UserPage.vue описує сторінку користувача. Він складається з компонентів User.vue і UserResults.vue, які описують загальну інформацію про користувача. Компонент UserResults.vue описує результати роботи користувача в системі. Тут описується кількість вправ, виконаних кожним кодеком, та список усіх виконаних тестів.

Компонент CodePage.vue описує вибрану сторінку кодека. Він містить загальну інформацію про вибраний кодек, а також три абстрактні динамічні компоненти: CodePractice.vue, CodeTheory.vue та CodeTest.vue. Коли вибрано певний код, абстрактний компонент замінюється конкретним компонентом. Навігація між компонентами здійснюється через вкладки за допомогою компонента Vuetify V-Tabs. Вкладені компоненти залишаються активними та не знищуються директивою keep-alive Vue. Це дозволяє користувачам вільно

переходити від теоретичної інформації до практичних завдань. Практична частина програми показана на рисунку 6.2.

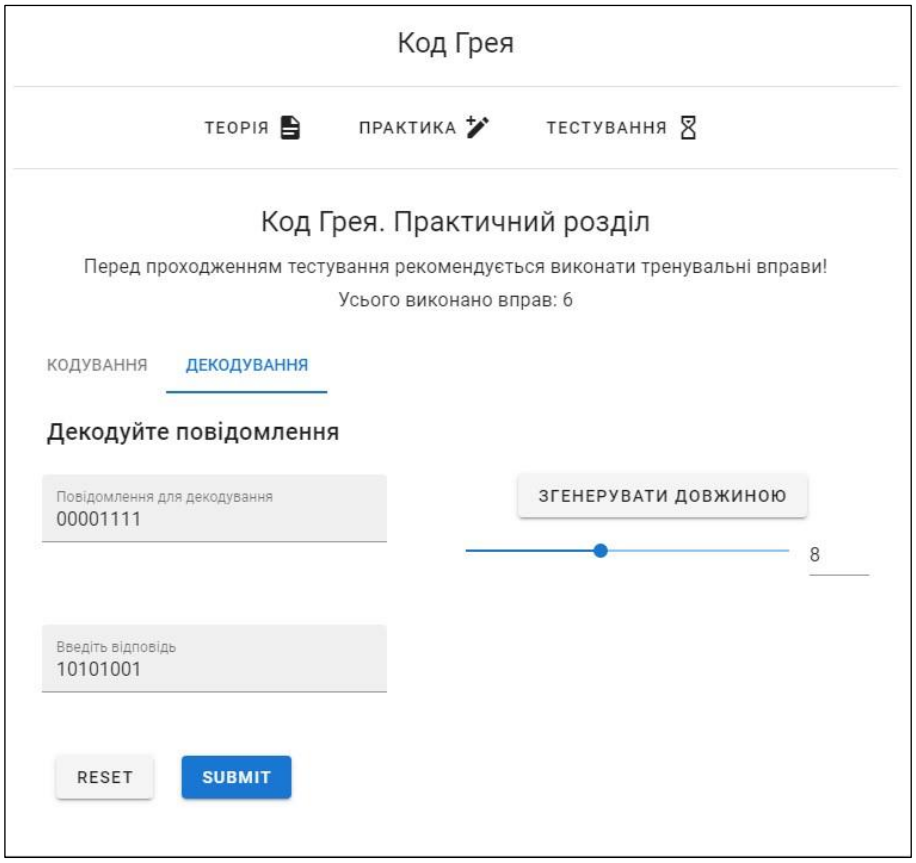

Рисунок 6.2 – Практичний розділ додатку (рисунок виконано самостійно)

Компонент InfoPage.vue описує сторінку довідки та надає додаткову інформацію користувачеві. Він надає поради та інструкції щодо використання системи, а також інформацію щодо розробки системи.

6.2 Маршрутизація додатку з використанням VueRouter

Ця система була розроблена з використанням VueRouter для створення багатосторінкового додатку. Ця частина екосистеми Vue дозволяє зіставити кожну URL-адресу з відповідною веб-сторінкою. Коли URL-адреса пошуку змінюється, VueRouter автоматично виконує дії для створення бажаного вигляду [29].

Налаштування маршрутизатора реалізовані у файлі router.js, який є частиною пакета маршрутизатора. Під час налаштування маршрутизатора обирається тип історії для обробки та встановлюється з'єднання з кореневим

екземпляром Vue. Шлях до програми, який буде використовуватися, описується і налаштовується в окремому файлі. При налаштуванні маршруту використовується об'єкт route, який приймає анонімний об'єкт з масивом маршрутів як параметр. Цей масив містить список об'єктів, що описують специфічні для програми шляхи. Кожен об'єкт має обов'язкові властивості: шлях для зберігання URL-адреси, ім'я шляху для швидкого доступу та компоненти для відображення при переході за цим шляхом. Зазвичай шлях вказується з компонентом, який описує всю веб-сторінку [30].

Для динамічного відображення VueRouter використовує компонент Router View, який автоматично вбудовує активний компонент шляху.

Оскільки система, яку ми розробляємо, невелика, але кількість гілок велика, було обрано декілька шляхів. Наступні шляхи використовуються для процесу авторизації:

- auth/register;
- auth/login;
- auth/logout.

Ці шляхи описують сторінки входу, авторизації та виходу відповідно. Сторінки входу і виходу описують шляхи входу і виходу, сторінки авторизації і виходу описують сторінки входу і виходу. Коли користувач виходить з системи, ця панель також має навігаційний захист, який запускається перед зміною панелі [31].

Навігаційні засоби захисту використовуються для перевірки стану програми та можливості переходу на потрібний маршрут. Одночасно вони перевіряють, чи користувач дійсно авторизований, і фіксують цей статус у Диспетчері. Якщо функція дозволена, викликається послідовність методів авторизації для виконання логіки авторизації. Якщо функціональність заборонена, користувач повертається до попереднього шляху, якщо це можливо, або відображається повідомлення про те, що для доступу необхідна авторизація. Такий захист навігації використовується для всіх шляхів, які пов'язані з функціями системи, такими як проходження тесту або перевірка оцінки. Тільки авторизовані користувачі мають

доступ до певних функцій системи. Захист навігації - це зручний спосіб обмежити доступ до вразливих частин програми.

Запустіть наступні панелі для ознайомлення з основними можливостями системи:

- /home;
- /info;
- /user/:id;
- /user/:id/results;
- /codes;
- /codes/:name.

Використовуйте панелі /home та /info для відображення домашньої та інформаційної сторінок програми відповідно. Вони використовують компоненти HomePage.vue та InfoPage.vue для відображення вмісту. Як і всі наступні панелі, вони використовують інструменти навігації для захисту конфіденційної інформації [32].

Для відображення сторінок користувачів пул /user/:id використовує елемент UserPage.vue. Він містить загальну інформацію про обліковий запис користувача, і користувач може редагувати особисту інформацію та налаштовувати свій профіль. Цей шлях також має підсторінку /user/:id/results, яка використовує елемент UserResults.vue. Використання підсторінки і компонента покращує продуктивність програми, оскільки змінюється лише частина веб-сторінки, а не вся сторінка. Цей компонент дозволяє користувачам переглядати результати тестування. Сторінка результатів користувача показана на рисунку 6.3.

Шлях /codes містить відповідний елемент CodeList.vue, який описує список усіх доступних кодів, представлений у вигляді маршрутизатора, спеціальний компонент, наданий VueRouter.Router-Link дозволяє вбудувати посилання на інший шлях як елемент у будь-який інтерфейс користувача.

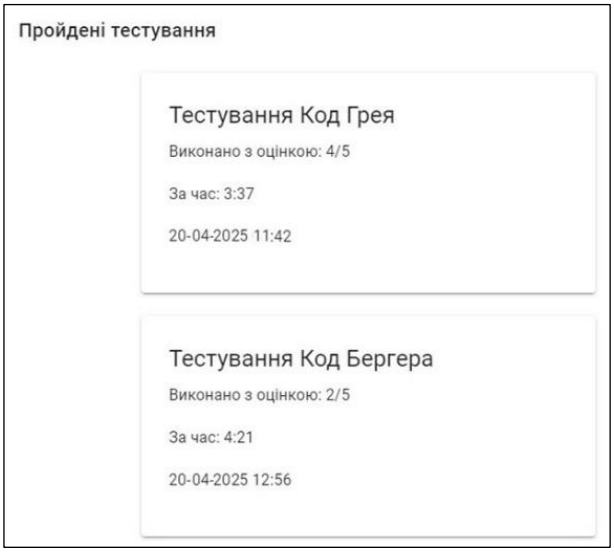

Рисунок 6.3 – Сторінка результатів користувача (рисунок виконано самостійно)

Router-Link використовується як основна ланка в системі завдяки багатьом можливостям налаштування. При виборі коду, наприклад, коду Бергера зі списку, Router-Link автоматично вибирає його назву зі списку кодів, доступних в системі, перевіряє, чи існує такий шлях, а потім перенаправляє користувача на шлях /codes/BergerCode, де користувачеві представлені сторінки та функції коду Бергера. Переваги VueRouter Перевага VueRouter полягає в тому, що динамічну маршрутизацію дуже легко налаштувати. Для коректного налаштування та відображення багатьох кодових сторінок використовується динамічний шлях /codes/:name для відображення абстрактного компонента Code.vue. Цей компонент складається з трьох підкомпонентів, які відображають функціональність коду: теоретичну інформацію про код, практичні завдання та тести, відповідно CodeTheory.vue та CodePractice.vue. Приклад компонента Tests показано на рисунку 6.4.

Ці компоненти містять лише прості користувацькі інтерфейси і замінюються спеціальними компонентами, коли ви переходите до динамічного шляху. Коли ви переходите до вибраного динамічного шляху, наприклад, /codes/BergerCode, VueRouter відображає компонент коду.

Під час компіляції Vue знаходить необхідні компоненти і компілює BergerTheory.vue, BergerPractice.vue і BergerTest.vue.

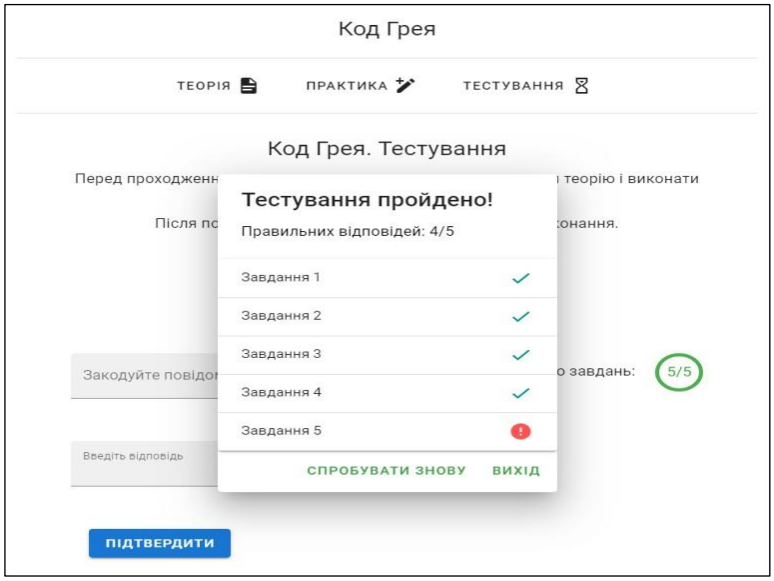

Рисунок 6.4 – Компонент тестування для коду Грея (рисунок виконано самостійно)

Цей розділ був присвячений реалізації користувацького інтерфейсу системи. Компоненти за замовчуванням були реалізовані за допомогою бібліотеки Vuetify з Material Design, а Vuetify було встановлено як плагін до системи. Інші залежності, такі як шрифти Google Roboto та іконки Material Design, були встановлені через CDN. Для створення необхідних сторінок використовуються компоненти «Розмітка», «Групування», “Форми” та «Кнопки». Поверх цих компонентів надбудовуються кастомні компоненти.

Компоненти вигляду програми поділяються на подання та набори компонентів. Подання описує компоненти, які представляють веб-сторінки додатка. Набори компонентів містять компоненти, які описують окремі елементи інтерфейсу, окремі форми, панелі навігації, меню. Вони використовуються для побудови веб-сторінок.

VueRouter використовується для маршрутизації односторінкових додатків. Використовується для створення переходів між елементами програми. Компоненти Vue пов'язані з URL-адресами. Коли викликається URL-адреса, VueRouter автоматично будує і відображає необхідні компоненти. Для розгалужених систем з великою кількістю вбудованих компонентів були створені підшляхи, які дозволяють здійснювати навігацію по різних частинах програми, не перевантажуючи основну частину.

ВИСНОВКИ

В рамках своєї кваліфікаційної роботи досліджено проблему вдосконалення методів дослідження роботи завадостійких кодеків. Результатом цієї роботи є створення системи моделювання, яка покращує навчальний процес їх дослідження, роблячи його більш доступним та зручним у використанні.

Для розробки цієї системи були проаналізовані аналогічні продукти. Як аналоги були обрані система навчання моделювання коду XTest+ та система моделювання та обробки сигналів SignalJ. Після аналізу були визначені основні проблеми, які має вирішити розроблена система. Найважливішими з них були: зберігання даних користувача, безпечний доступ до системи та інтерфейс для налаштування процесу моделювання.

З урахуванням недоліків аналізованих аналогів були визначені функціональні та нефункціональні вимоги до системи. Основні функціональні вимоги полягають у тому, щоб система дозволяла зберігати дані користувача та результати роботи в додатку. Крім того, були визначені вимоги до функціональності системи: надання теоретичної інформації користувачам, можливість виконання практичних завдань та проходження тестів. Нефункціональні вимоги стосуються якості та продуктивності системи, швидкості генерації завдань, точної оцінки результатів тощо.

Випадки використання визначаються під час розробки системи. Як неавторизовані, так і авторизовані користувачі визначаються як ключові зацікавлені сторони. Неавторизовані користувачі мають доступ до функцій системи лише після відповідної авторизації. Авторизовані користувачі можуть вільно взаємодіяти з програмою, переглядати теоретичну інформацію, виконувати практичні завдання та переглядати результати та досягнення.

Для впровадження цієї системи були проаналізовані існуючі технології та вибрали найбільш підходящі. Виходячи з вимог, ми вирішили реалізувати систему як безсерверний односторінковий додаток з використанням Firebase. Частина логіки авторизації та управління базою даних була передана на аутсорсинг. Для клієнтської реалізації ми використовували веб-фреймворк Vue.js, оскільки він

найкраще підходив для цих завдань. Також використоні основні компоненти екосистеми Vue: VueRouter, VueResource, Vuex та Vuetify. Основна розробка виконується мовою програмування JavaScript.

На основі обраних технологій розроблено схему архітектури системи, що складається з двох основних частин: сервісу Firebase, що діє як сервер, та односторінкового додатку Vue. Додаток має компонентну архітектуру, розділену на незалежні модулі. Основні модулі включають модуль для зв'язку з Firebase, модуль з логікою менеджера станів та три модулі бізнес-логіки для тестування, користувацького та кодекового функціоналу. Останній модуль містить компоненти презентації користувацького інтерфейсу. У систему також інтегровані сторонні пакети, такі як бібліотека Vuetify та інші частини екосистеми Vue.

Система використовує нереляційну базу даних Firebase для зберігання даних у форматі JSON та оновлюється в режимі реального часу через синхронізацію з клієнтом. Було розроблено ER-діаграму бази даних для визначення сутностей та зв'язків між ними, необхідних для впровадження системи. Основними визначеними сутностями є: «Користувач» – профіль користувача; «Завершений тест» – завершений тест; «Активний тест» – активний тест; «Журнал вправ» – огляд виконаних завдань вправ; та «Тестове завдання» – тестове завдання.

Більша частина бізнес-логіки реалізована на стороні клієнта. Додаткова безпека системи забезпечується шляхом міграції авторизації та бази даних до сервісів Firebase. Основна мета системи – дозволити користувачам вивчити, як працює кодек, ознайомитися з теоретичною інформацією, виконати практичні завдання та пройти тестування. Всі результати використання системи користувачами зберігаються в базі даних. Бізнес-логіка реалізована за допомогою шаблону MVVM, який реалізований у всіх компонентах Vue.js. Також використовувалися інші методи розробки: розподіл стану програми через обробник Vuex, розділення логіки на окремі файли та об'єднання цих файлів у компоненти повторного використання.

Інтерфейс користувача розроблено за допомогою бібліотеки Vuetify. Бібліотека надає велику кількість попередньо створених компонентів для

елементів інтерфейсу користувача. Розроблені компоненти за допомогою Vuetify, яка поділена на пакети перегляду, що містять цілі веб-сторінки, та компоненти, що є окремими елементами цих веб-сторінок. VueRouter використовується для маршрутизації програми. Під час налаштування шляху кожен URL-шлях відповідає компоненту веб-сторінки. Це дозволяє користувачеві переміщатися між різними частинами програми.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Полторак В.П. Теорія інформації та кодування / В.П. Полторак, Ю.П. Жураковський // Підручник. – К.: Вища школа, 2001. – 255 с.

2. Полторак В.П. Калькулятор GF(q) для циклічних кодів / В.П. Полторак, Н.С. Вітіщенко // Вісник НТУУ «КПІ». Інформатика, управління та обчислювальна техніка. Зб. наук. пр. Вип. 53. – К.: Век+, 2011. – С. 233–240.

3. ДСТУ 9241-11:2006 Ергономічні вимоги до роботи з відеотерміналами в офісі. Частина 11. Настанови щодо прийнятності у використанні.

4. Забезпечення якості. Функціональні та нефункціональні вимоги. – Доступ: http://lvivqaclub.blogspot.com/2008/10/blog-post_17.html

5. Functional VS Non-functional Requirements (With Examples) – Доступ:https://visuresolutions.com/requirements-management-traceability-guide/ functional-vs-non-functional-requirements/

6. Як та навіщо використовувати Use Cases – Доступ: https://dou.ua/ lenta/articles/use-cases/

7. Desktop Application vs Mobile App vs Web App – Доступ: https://www.miraclemobile.io/desktop-apps-vs-web-apps-vs-mobile-apps/

8. Desktop vs mobile: user experience optimization – Доступ: https://99designs.com/blog/web-digital/desktop-vs-mobile-app-design/

9. Web and Desktop Usage In 2019 – Доступ: https://www.perficientdigital.com/insights/our-research/mobile-and-desktop-usagestudy

10. Web Application Architecture: How the Web Works – Доступ: https://www.altexsoft.com/blog/engineering/web-application-architecture-how-theweb-works/

11. How Single-Page Applications work – Доступ: https://blog.pshrmn.com/entry/how-single-page-applications-work/

12. jQuery Official API documentation – Доступ: https://api.jquery.com/

13. Чому я досі використовую jQuery у 2019 році – Доступ: https://webformyself.com/pochemu-ya-do-six-por-ispolzuyu-jquery-v-2019-godu/

14. Developer Survey Results 2018 – Доступ: https://insights.stackoverflow.com/survey/2018

15. Angular: Best Use Cases and Reasons To Opt For This Tool – Доступ: https://yalantis.com/blog/when-to-use-angular/

16. 7 Reasons why you should use React – Доступ: https://stories.jotform.com/7-reasons-why-you-should-use-react-ad420c634247

17. Introduction to Vue.js 2.0 – Доступ: https://vuejs.org/v2/guide/index.html

18. Vue.js with TypeScript – Доступ: https://johnpapa.net/vuetypescript/

19. Мова програмування JavaScript – Доступ: https://developer.mozilla.org/ en-US/docs/Web/JavaScript

20. Uses of Firebase apps – Доступ: https://firebase.google.com/ products/#develop-products

21. The Best Material Design Vue Framework – Доступ: https:// medium.com/@johnmaeda/the-best-material-design-vue-js-frameworkbd7e2b730b5a

22. Why Vuetify – Доступ: https://vuetifyjs.com/en/introduction/why-vuetify

23. Working with Webpack – Доступ: https://cli.vuejs.org/guide/ webpack.html

24. Structure your data in Firebase – Доступ: https://www.airpair.com/ firebase/posts/structuring-your-firebase-data

25. Structuring a Vue project – Доступ: https://medium.com/@zitko/ structuring-a-vue-project-87032e5bfe16

26. Design Patterns: Elements of Reusable Object-Oriented Software / Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. – Addison-Wesley Professional, 1995. – 396 p.

27. Vue.js MVVM Concept Overview – Доступ: https://012.vuejs.org/ guide/mvvm

28. Vue architecture patterns – Доступ: https://medium.com/@ederng/ the-vue-architecture-that-worked-for-me-in-small-andlarge-apps-9b253cf92951

29. Vue Router Documentation – Доступ: https://router.vuejs.org/

30. Neural Network Architecture Editor With Code Generation / Maksym Bekuzarov; Mariya Shirokopetleva; Oleksandr Samantsov; Oksana Mazurova // Problem of Infocommunications. Science and Technology (PIC S&T′2020), Kharkiv, Ukraine.- 6-9 October 2020. – Pp. 701-706.

31. Finite predicate-driven logic networks method for enhanced education data analysis / Dudar Z., Kozyriev A. // Radioelectronic and Computer Systems. 2024. Vol. 2024, no. 3. P. 205–215 https://doi.org/10.32620/reks.2024.3.14

32. Convolutional neural network scaling methods in semantic segmentation / Hmyria, I. O., Kravets, N. S. // Radio Electronics, Computer Science, Control, 2024(2), Pp. 52-60 https://doi.org/10.15588/1607-3274-2024-2-6

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ ЗА НАУКОВИМИ НАПРЯМАМИ КЕРІВНИКА ТА НАУКОВЦІВ КАФЕДРИ ПРОГРАМНОЇ ІНЖЕНЕРІЇ

30. Neural Network Architecture Editor With Code Generation / Maksym Bekuzarov; Mariya Shirokopetleva; Oleksandr Samantsov; Oksana Mazurova // Problem of Infocommunications. Science and Technology (PIC S&T′2020), Kharkiv, Ukraine.- 6-9 October 2020. – Pp. 701-706.

31. Finite predicate-driven logic networks method for enhanced education data analysis / Dudar Z., Kozyriev A. // Radioelectronic and Computer Systems. 2024. Vol. 2024, no. 3. P. 205–215 https://doi.org/10.32620/reks.2024.3.14

32. Convolutional neural network scaling methods in semantic segmentation / Hmyria, I. O., Kravets, N. S. // Radio Electronics, Computer Science, Control, 2024(2), Pp. 52-60 https://doi.org/10.15588/1607-3274-2024-2-6