ЕКОНОМІКО-ТЕХНОЛОГІЧНИЙ ІНСТИТУТ ІМЕНІ РОБЕРТА ЕЛЬВОРТІ
КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

«Допущено до захисту»
Завідувач кафедри
інформаційних технологій
_____Бондар О.П.
«19» 06 2024 р.

Кваліфікаційна робота

на здобуття ступеня вищої освіти «бакалавр» зі спеціальності 122 «Комп'ютерні науки»

на тему:

«Розробка інтерактивного посібника на основі JavaScript-фреймворку Electron»

Анастасьєва Анастасія Євгенівна

Керівник кваліфікаційної роботи:
Сурков Костянтин Юрійович,
кандидат технічних наук

Роботу рекомендовано до захисту
на засіданні кафедри інформаційних технологій
Протокол №10 від «06» 06 2024 р.
Завідувач кафедри інформаційних технологій
_____ Бондар О.П.

Роботу захищено на засіданні ЕК
з оцінкою
Відмінно / A / 95
(за національною шкалою, шкалою ECTS, бали)

Протокол №8 від «20» 06 2024 р.
Голова ЕК _____

Кропивницький – 2024

АНОТАЦІЯ

Анастасьєва А.Є. Розробка інтерактивного посібника на основі JavaScript-фреймворку Electron – Кваліфікаційна робота зі спеціальності 122 «Комп'ютерні науки». – Економіко-технологічний інститут імені Роберта Ельворті, Кропивницький, 2024.

Кваліфікаційна робота складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків.

Загальний обсяг роботи становить 94 сторінок, включаючи 46 рисунків, 20 літературних джерел та 3 додатки.

Метою даної роботи є розробка інтерактивного посібника для навчання веброзробці з використанням JavaScript-фреймворку Electron, який забезпечує зручний доступ до навчального контенту, можливість проходження тестів, створення нотаток та персоналізований досвід для користувачів.

Об'єкт дослідження: процес розробки інтерактивного посібника з використанням сучасних вебтехнологій.

Предмет дослідження: інтерактивний посібник, розроблений на основі JavaScript-фреймворку Electron, з функціями авторизації та збереження даних у базі даних Firebase.

Основні результати роботи: проаналізовано існуючі рішення та обрано технології для реалізації посібника, розроблено структуру та архітектуру застосунку, що включає основний процес, рендерний процес та інтерпроцесну комунікацію (IPC), реалізовано систему реєстрації та автентифікації користувачів з використанням Firebase Authentication, створено особистий кабінет користувача для відображення процесу навчання, історії тестів та нотаток, розроблено та впроваджено структурований навчальний контент з теоретичними матеріалами, прикладами коду та практичними завданнями, реалізовано систему тестувань знань користувачів з аналізом допущених помилок, проведено тестування всіх функцій застосунку та забезпечено безперебійну роботу.

Ключові слова: застосунок, JavaScript-фреймворк Electron, база даних, авторизація.

ЗМІСТ

ВСТУП

Сучасний світ цифрових технологій вимагає від освітніх ресурсів адаптації до постійно змінюваних потреб користувачів, які шукають ефективні та гнучкі способи навчання. Важливість технологічно орієнтованого освітнього контенту, що сприяє самостійному навчанню, особливо актуалізується у контексті швидкого розвитку програмування та веброзробки.

Зростальний попит на персоналізоване навчання, яке б включало інтерактивні елементи та було доступне україномовним користувачам, посилює актуальність розробки інтерактивного посібника на основі JavaScript-фреймворку Electron. Розробка ведеться з урахуванням локальних особливостей та можливостей доступу до навчальних ресурсів частково офлайн, що є важливим для регіонів з обмеженим доступом до Інтернету. Застосунок дозволить користувачам не тільки навчатися нових технологій, але й відстежувати свій прогрес, аналізувати помилки, зберігати нотатки та персоналізувати досвід використання через систему налаштувань.

Метою кваліфікаційної роботи є створення інтерактивного посібника на основі JavaScript-фреймворку Electron, який інтегрує теоретичні знання та практичні завдання у єдиний інтуїтивно зрозумілий інтерфейс. Посібник розроблено на основі Electron для забезпечення кросплатформної сумісності, а для функціонала авторизації та збереження даних користувачів інтегровано Firebase. Основні завдання роботи включають: розробку структурованого навчального контенту, створення інтерактивних модулів для тестових завдань, інтеграцію системи авторизації та збереження прогресу користувачів, а також реалізацію адаптивності інтерфейсу для різних платформ.

Практичне значення роботи полягає в створенні кросплатформного застосунку, який дозволяє користувачам не тільки вивчати матеріал, але й безпосередньо застосовувати отримані знання на практиці через завдання та тести. Це сприяє кращому засвоєнню матеріалу та розвитку практичних навичок програмування.

На сьогодні більшість онлайн-платформ для вивчення програмування не забезпечують достатньої інтерактивності, глибини матеріалу та функціональності офлайн-доступу. Проблема також полягає у відсутності локалізованих українськомовних ресурсів, що робить процес навчання складнішим для вітчизняних користувачів.

Об'єктом проєктування є процес розробки інтерактивного посібника з веброзробки, а предметом – інтерактивний посібник на основі Electron. Основними методами реалізації є програмування на JavaScript, використання Electron, проєктування користувацького інтерфейсу та інтеграція з хмарними сервісами Firebase.

Таким чином, дана кваліфікаційна робота спрямована на розв’язання актуальної проблеми доступу до якісних та комплексних освітніх ресурсів з веброзробки для україномовних користувачів, з урахуванням можливостей сучасних технологій.

РОЗДІЛ 1. ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Теоретичні відомості про застосунки на основі фреймворку Electron

Розвиток настільних застосунків з кожним роком зростає, і з’являється все більше вдосконалень та оновлень щодо системи, що працюють у Linux, Mac, Windows, це вимагає постійного обслуговування, щоб підтримувати програми в актуальному стані [5]. Для того, щоб отримати найкращу версію застосунків для кросплатформної роботи десктопних систем, Electron є гарним рішенням.

Electron – це фреймворк відкритим кодом, який дозволяє створювати настільні програми використовуючи вебтехнології такі як: HTML, CSS, JavaScript. З появою цього фреймворку з’явилася можливість створювати повнофункціональні кросплатформні застосунки використовуючи знайомі вебтехнології. Варто зауважити, що Electron не обмежується лише вебстеком. Зокрема, Electron використовує Node.js та вбудоване API для своєї серверної частини, завдяки цьому розробники мають доступ до низькорівневих системних функцій та можливостей операційної системи. HTML та CSS використовується для клієнтської частини, а Chromium для зовнішньої частини.

Гнучкість фреймворку дозволяє створювати застосунки, які виглядають і працюють як нативні, забезпечуючи той самий рівень інтеграції та користувацького досвіду, що і традиційні настільні програми. Наприклад застосунки Electron можуть мати власні меню, панелі інструментів, контекстні меню та навіть системні сповіщення, повністю інтегруючись в середовище операційної системи.

Цей фреймворк був розвинений компанією GitHub та вперше представлений у 2013 році під назвою AtomShell, потім перейменували в Electron.

Крім того, Electron дозволяє розробникам створювати багатовіконні застосунки, керувати їх життєвим циклом, налаштовувати параметри вікон. Це

відкриває нові можливості для створення складних і гнучких користувацьких інтерфейсів.

Electron має велику перевагу в створенні десктопних застосунків, і ця перевага полягає в тому, що Electron Apps можна запускати на більшості популярних операційних систем: MacOS, Linux, Windows [19]. Зазвичай доводиться кодувати одну програму для кожної системи, і це забирає багато часу на розробку, адже потрібно вивчати багато різних мов, опановувати досконало як працює та чи інша операційна система, а також оптимізувати візуальні елементи та макети, щоб забезпечити відповідність кожній системі.

Ще одна перевага Electron в тому, що він автоматично робить програму адаптивною, щоб вона відповідала будь-якому екрану, незалежно від того який ви використовуєте: великий чи маленький. Крім того, все закодовано в одній кодові базі.

Фреймворк Electron має таку архітектуру, яка складається з двох частин: головного процесу (Main Process) та процесу рендерингу (Render Process) (див. рис. 1.1). Головний процес управляє життєвим циклом застосунку, створює вікна та взаємодіє з системними ресурсами через вбудований вебрушій Chromium. Процес рендерингу відповідає за відображення вебконтенту та виконання JavaScript коду в середині вікон застосунку, використовуючи таку саму архітектуру, що і сучасні браузери.

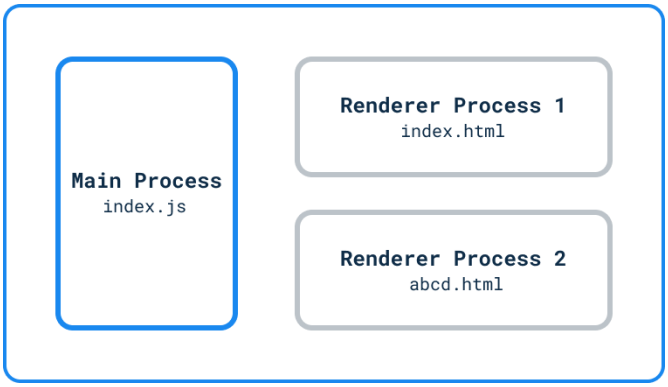


Рис. 1.1. Приклад архітектури застосунку на Electron

Electron забезпечує потужну інтеграцію з операційною системою завдяки власному міжпроцесорному механізму обміну повідомлення (IPC – Inter-Process Communication). Через цей механізм можна отримати доступ до

широкого спектра системних API, що відкриває численні можливості для взаємодії застосунків з різними системними компонентами та ресурсами.

Серед основних API, доступних в Electron, можна виділити наступні:

1. Взаємодія з файловою системою:застосунки можуть читати, записувати, переміщувати та видаляти файли та директорії на локальному пристрої, забезпечуючи повноцінну роботу з файловою структурою.

2. Мережеві операції: API Electron дозволяють застосункам здійснювати HTTP-запити, керувати проксі-серверами, працювати з WebSocket-з'єднаннями та виконувати інші мережеві операції.

3. Робота з меню та сповіщеннями: можна створювати нативні меню, контекстні меню, панелі завдань та системні сповіщення, що забезпечує інтуїтивну взаємодію застосунків з користувачами.

4. Діалогові вікна: Electron надає API для створення стандартних діалогових вікон операційної системи, таких як вікна для відкриття/збереження файлів, повідомлення про помилки та попередження.

5. Управління вікнами: застосунки можуть створювати, керувати та налаштовувати вікна з різними параметрами, такими як прозорість, позиція, розміри, стан (мінімізоване, максимізоване).

6. Інтеграція з робочим столом: API Electron дозволяють застосункам взаємодіяти з робочим столом, створювати ярлики, працювати з буфером обміну даними та виконувати інші операції на рівні робочого столу.

Крім численних системних API, Electron надає корисні інструменти для полегшення розробки та розповсюдження застосунків:

- Автоматичне оновлення: вбудована підтримка автоматичного оновлення застосунків, що дозволяє користувачам завжди мати найновішу версію програми без необхідності вручну перевстановлювати її.

- Створення інсталяторів: фреймворк надає інструменти для створення нативних інсталяторів для різних операційних систем, таких як .exe для Windows, .dmg для macOS та .deb/.rpm для Linux.

- Пакування застосунків: можливість пакувати застосунки у портативні автономні пакети, які можна легко розповсюджувати та. Запускати на різних платформах без необхідності встановлення.
- Налаштування іконок та метаданих: розробники можуть налаштовувати іконки, назви та інші метадані застосунків для кожної цільової платформи, забезпечуючи нативний вигляд програми.

Ця широка підтримка системних API та додаткових інструментів ще раз підкреслює, що Electron є потужною платформою для створення різних десктопних застосунків, що можуть повноцінно інтегруватися з ОС та забезпечувати багатий користувацький досвід, характерний для нативних програм.

Розробка та налагодження Electron застосунків можуть бути досить не простими та складними, оскільки вони поєднують вебтехнології з нативними компонентами. Тому важливо мати належні інструменти для ефективного виявлення та усунення проблем. Electron пропонує кілька вбудованих інструментів налагодження, а також підтримує використання популярних засобів для веброзробки.

Одним з найбільш корисних є Інспектор Electron, який дозволяє переглядати та редагувати DOM, відстежувати мережеві запити, налагоджувати JavaScript та взаємодіяти з Node.js API.

Для профілювання продуктивності та виявлення вузьких місць у коді можна використовувати стандартні інструменти Node.js, такі як console.profile та console.profileEnd, а також сторонні профілювальники, наприклад, @electron/remote та @hirojith/electron-memwatcher. Ці інструменти допомагають виявляти проблеми з пам’яттю, застигання та вузькі місця в CPU.

Крім того, фреймворк підтримує популярні для налагодження вебзастосунків, такі як React Developer Tools та Redux DevTools. Ці інструменти дозволяють інспектувати стан застосунку, відстежувати зміни стану та налагоджувати компоненти, що особливо корисно при використанні фреймворків, таких як React чи Angular.

Безпека та продуктивність є ключовими аспектами при розробці Electron застосунків. Оскільки Electron вбудовує вебтехнології в нативні застосунки, це може створювати певні ризики безпеки та проблеми з продуктивністю, якщо вживати належних заходів.

Потрібно врахувати рекомендації та практики забезпечення безпеки Electron застосунків. Потрібно не забувати регулярно оновлювати фреймворк до останніх стабільних версій для виправлення вразливостей. Також варто обмежувати доступ та дозволи для Node.js API відповідно до принципу мінімальних привілеїв. Варто використовувати контекст ізоляції для процесів рендерингу, щоб запобігти ін’єкціям коду на стороні клієнта. Не забувати використовувати авторитетні бібліотеки та модулі з активною підтримкою та регулярними оновленнями безпеки.

Задля оптимізації продуктивності та зменшення споживання ресурсів потрібно уникати непотрібних операцій та ресурсомістких обчислень на головному процесі [3]. Потрібно використовувати методи ефективного керування пам’яттю, такі як правильне видалення подій та циклів EventLoop.

Тестування є невіддільною частиною розробки якісних та надійних Electron застосунків. Тестування допомагає виявити та усувати помилки на ранніх стадіях розробки, забезпечуючи стабільність та очікувану поведінку застосунків на різних платформах.

Існують різні підходи до тестування Electron застосунків. Першим відомим підходом є модульне тестування – це тестування окремих компонентів, функцій та модулів на рівні коду. Це дозволяє ізольовано перевірити правильність їх роботи. А от інтеграційне тестування – це тестування взаємодії між різними компонентами та модулями, забезпечуючи, що вони працюють разом як єдине ціле. І ще один відомий підхід є приймальне тестування – це кінцеве тестування застосунків на різних платформах та конфігураціях, імітуючи реальне використання користувачами.

Популярними фреймворками та бібліотеками для тестування Electron застосунків вважаються:

- Jest: потужний фреймворк для модульного тестування JavaScript, який добре інтегрується з Electron.
- Spectron: бібліотека для функціонального тестування Electron застосунків, що надає високорівневий API для управління застосунками та взаємодії з користувацьким інтерфейсом.
- WebDriverIO: фреймворк для автоматизованого тестування вебзастосунків, який також підходить для тестування Electron застосунків.

Ефективне тестування Electron застосунків передбачає використання різних підходів та інструментів на різних рівнях, від модульного тестування окремих компонентів до повноцінного функціонального тестування всього застосунку на різних платформах. Це забезпечує високу якість та надійність застосунків, а також полегшує виявлення та усунення помилок на ранніх стадіях розробки.

Завдяки своїй широкій функціональності та інтеграції з вебтехнологіями, Electron став популярною платформою для розробки різноманітних настільних застосунків, включаючи текстові редактори, клієнти електронної пошти, музичні плеєри, IDE для програмування та багато іншого. Відомі проєкти, засновані на Electron, включають Discord, Figma, Dropbox, GitHub Desktop, Loom, MongoDB Compass, Notion, Signal, Skype, Visual Studio Code, Whatsapp Desktop, Microsoft Teams, Trello, Twitch.

Настільна програма WebTorrent також створена за допомогою Electron та інших інструментів JavaScript і використовує технологію WebRTC для одноранговогo підключення. WebTorrent Desktop створено для основних операційних систем, таких як MacOS, Linux, Windows. Він підключає програму до всіх популярних мереж BitTorrent і WebTorent.

Популярна система керування вмістом WordPress, яка використовується для редагування та керування вебсайтом. Але окрім вебверсії, десктопна програма також була створена з використанням Electron для надсилання на кількох платформах одночасно. Тож користувачі MacOS або windows можуть використовувати робочий стіл WordPress без будь-яких відволікань і проблем

використовуючи функції WordPress повною мірою. За допомогою фреймворку застосунок WordPress для робочого столу швидко завантажується на всіх платформах. Окрім Electron, бібліотека react також була використана для надання захопливого досвіду користувача.

Корпоративний менеджер Slack, який використовується серед співробітників компаній будь-якого розміру теж розроблений на основі JavaScript фреймворку Electron. Настільний застосунок доступний для всіх популярних операційних систем: Linux, macOS, Windows, а також мобільні версії для IOS та Android.

Ще однією з переваг Electron є можливість розробляти застосунки з використанням популярних фреймворків та бібліотек JavaScript, таких як React, Angular чи Vue.js. Це дозволяє командам розробників використовувати наявні знання та навички веброзробки, а також сприяє кращій підтримці та спрощеному обслуговуванню коду завдяки повторному використанню наявних рішень та інструментів.

Важливо зазначити, що Electron надає розробникам величезну гнучкість щодо оформлення та поведінки застосунків. На відміну від вебзастосунків, де оформлення та взаємодія переважно визначаються браузером, у застосунках на Electron розробники мають повний контроль над візуальними елементами та користувацьким досвідом. Це дозволяє створювати застосунки з нативним відчуттям та поведінкою, характерною для конкретної операційної системи.

Отже, JavaScript фреймворк Electron добре підходить для створення кросплатформних настільних програм за допомогою вебтехнологій. Він пропонує ряд значних переваг, таких як: доступ до нативних можливостей, можливість повторного використання коду, багату екосистему, швидке створення прототипів і широке впровадження.

1.2. Аналіз наявних аналогічних продуктів

Аналіз наявних рішень в галузі є критично важливим етапом при розробці будь-якого нового програмного продукту. Ретельне вивчення наявних альтернатив дозволяє отримати цілісну картину поточного стану

речей, виявити потенційні прогалини та невикористані можливості на ринку. Саме тому ґрунтовний аналіз аналогічних продуктів є невіддільною складовою процесу розробки інтерактивного посібника на основі JavaScript фреймворку Electron.

Також дуже важливим є SWOT-аналіз, тому що він допоможе зрозуміти ринок аналогічних застосунків в цілому, а також за допомогою нього буде легко систематизувати інформацію про сильні та слабкі сторони, можливості та загрози вже наявних продуктів. Для кожного продукту окремо буде визначено:

- Сильні сторони (Strengths): тут прописуються унікальні функції які роблять продукт кращим та зручнішим;
- Слабкі сторони (Weaknesses): прописуються недоліки чи основні аспекти які відштовхують користувачів від продукту;
- Можливості (Opportunities): описуються фактори які сприятимуть розвитку продукту;
- Загрози (Threats): описуються ризики які негативно впливають на успішність продукту

Під час дослідження було виявлено низку наявних аналогів, які можна розділити на кілька основних категорій: вебзастосунки та онлайн-платформи, мобільні застосунки та настільні програми. Кожна з цих категорій має свої унікальні особливості, переваги та недоліки.

Варто зазначити, що аналіз проводився з урахуванням не лише світового ринку програмних продуктів, але й з особливою увагою до ринку України. Результати пошуку засвідчили відсутність прямих аналогів інтерактивних навчальних посібників з комплексним функціоналом на українському ринку. Це підкреслює актуальність та новизну розробленого програмного забезпечення для потенційних користувачів в Україні.

Learn Git Branching - це інтерактивний вебзастосунок, спеціально розроблений для візуального вивчення системи контролю версій Git.

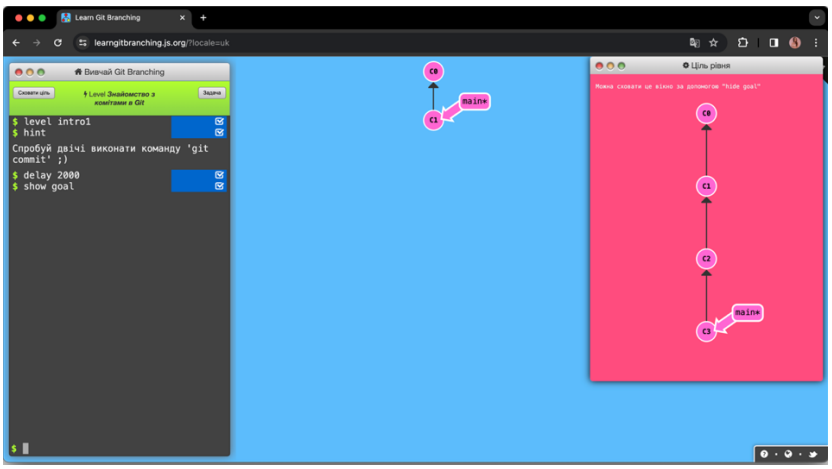


Рис. 1.2. Приклад проходження завдання в Learn Git Branching

Його ключовою особливістю є серія інтерактивних завдань, під час яких користувачі можуть практикуватися у виконанні різних команд Git, спостерігаючи за динамічними змінами у візуалізованому репозиторії в режимі реального часу. Вебзастосунок також пропонує теоретичні навчальні матеріали, підказки та пояснення для кращого розуміння концепцій.

Learn Git Branching розроблений з використанням JavaScript та візуалізації на основі бібліотеки D3.js. Він має модульну архітектуру, що полегшує розширення та підтримку. Однак код застосунку закритий і не підтримується активно розробниками.

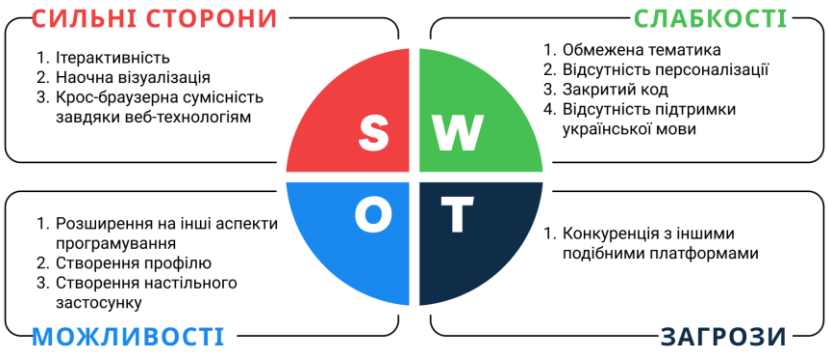


Рис. 1.3. SWOT-аналіз для Learn Git Branching

Безперечними перевагами Learn Git Branching є його інтерактивність, зручна візуалізація процесів Git та акцент виключно на цій специфічній темі, що дозволяє користувачам зосередитися та поглибити свої знання.

З недоліків варто відзначити обмеженість лише темою Git, що не охоплює інші аспекти програмування та розробки програмного забезпечення, а також відсутність особистого кабінету для відстеження прогресу навчання.

Codecademy - популярна онлайн-платформа для інтерактивного навчання програмування, розроблена з використанням Ruby on Rails, React.js та Redux. Курси складаються з теоретичних матеріалів та інтерактивних вправ, під час яких користувачі можуть відразу застосовувати отримані знання на практиці у вбудованому інтегрованому середовищі розробки (IDE) прямо у браузері.

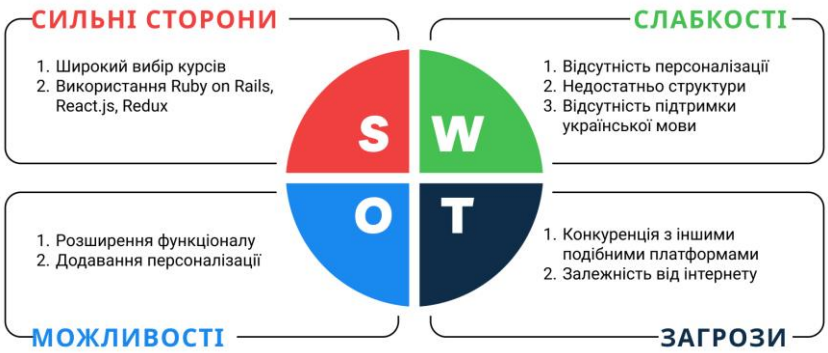


Рис. 1.4. SWOT-аналіз для Codecademy

Codecademy вирізняється широким вибором курсів, чіткими інструкціями та можливістю отримувати зворотний зв'язок.

Однак великим недоліком є брак структурованих навчальних матеріалів, відсутність повноцінного персоналізованого середовища для експериментів та розробки реальних проєктів поза обмеженим середовищем вправ. Крім того, хоча Codecademy має застосунок для настільних пристроїв, його функціонал дуже обмежений, оскільки сам процес навчання проходить на вебсайті, а не в застосунку. Також недоліком є те, що в застосунку особистий кабінет обмежений інформацією, оскільки аналізу тестувань чи практичних завдань не має, є тільки можливість відстежувати тривалість навчання.

SoloLearn - мобільний застосунок, який розроблений на Flutter та Dart для вивчення програмування. Він пропонує інтерактивні вправи та завдання для різних мов програмування.

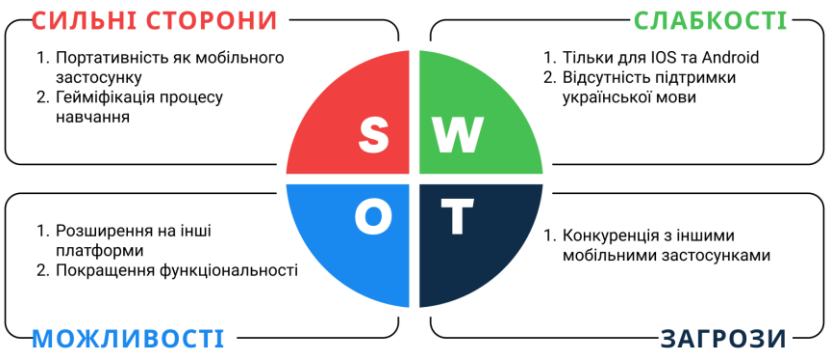


Рис. 1.5. SWOT-аналіз для SoloLearn

Основними перевагами SoloLearn є портативність, гейміфікація процесу навчання та спільнота однодумців. Однак, як мобільний застосунок, він може бути обмежений у функціональності та не надавати повноцінного середовища розробки. Ще один недолік застосунку SoloLearn для українського ринку – це відсутність підтримки української мови в цьому застосунку.

Code: Learn Coding/Programming - ще один мобільний застосунок для iOS та Android, орієнтований на спрощене вивчення основ програмування. Він пропонує пояснення концепцій та базових прикладів коду.

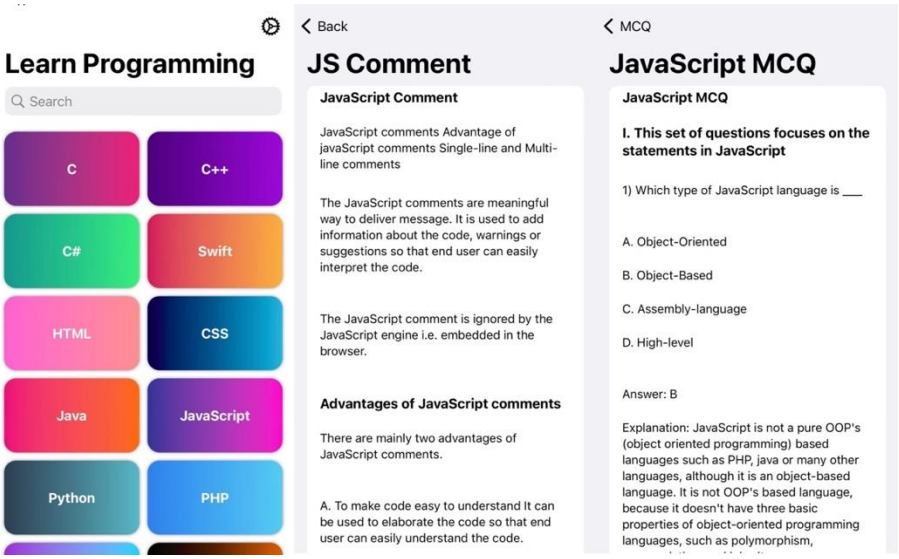


Рис. 1.6. Приклад як виглядає мобільний застосунок Code: Learn Coding/Programming

Застосунок складається тільки з теоретичного матеріалу, а тести не мають перевірку знань як користувач освоїв матеріал, тому що подані одразу з готовими відповідями.

Застосунок Code: Learn Coding/Programming розроблений з використанням React Native та Redux. Продукт відрізняється простим та інтуїтивним інтерфейсом, що робить процес навчання більш доступним для користувачів без попереднього досвіду.

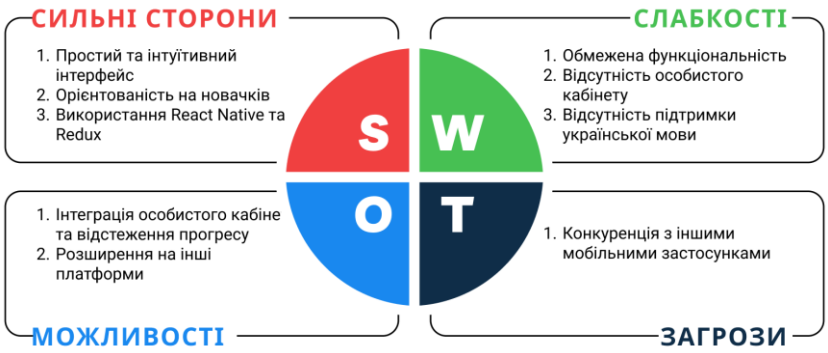


Рис. 1.7. SWOT-аналіз для Code: Learn Coding/Programming

Безперечною перевагою Code: Learn Coding/Programming є його зрозумілість та орієнтованість на новачків, що дозволяє легко долучитися до світу програмування.

Проте недоліком є більший акцент на теоретичних аспектах порівняно з практичними вправами, а також відсутність особистого кабінету та системи відстеження прогресу навчання.

Загалом, мобільні застосунки мають переваги у вигляді портативності та зручності використання. Однак вони часто обмежені функціоналом та не можуть повністю замінити повноцінне настільне середовище розробки для поглибленого навчання програмування.

Programming Hub - це настільний застосунок для вивчення програмування з інтерактивними вправами, розроблений з використанням фреймворку Electron та мови програмування JavaScript. Він пропонує ігровий підхід до навчання, де користувачі проходять рівні та отримають нагороди.

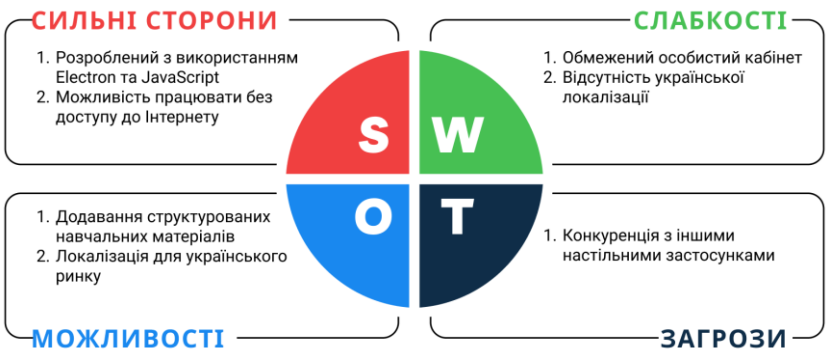


Рис. 1.8. SWOT-аналіз для Programming Hub

Перевагами Programming Hub є захопливий ігровий формат, що мотивує користувачів продовжувати навчання, можливість працювати без доступу до Інтернету та кросплатформеність завдяки використанню Electron.

Однак недоліками можна вважати відсутність детальних пояснень та структурованих навчальних матеріалів, обмеженість тематики виключно програмуванням, а також відсутність української локалізації. В застосунку є особистий кабінет, але обмежений різноманітними функціями. В кабінеті користувач може тільки відстежувати які курси він закінчив.

Java Recipes - це застосунок, який містить каталог прикладів коду для Java та SQL запитів. Він орієнтований на надання "рецептів" коду для вирішення типових завдань. Розроблений з використанням Objective-C та Swift для iOS та MacOS відповідно. Він містить каталог прикладів коду, відібраних з книг та Інтернет-ресурсів, що можуть бути корисними для розв’язання поширених проблем або підготовки до співбесід.

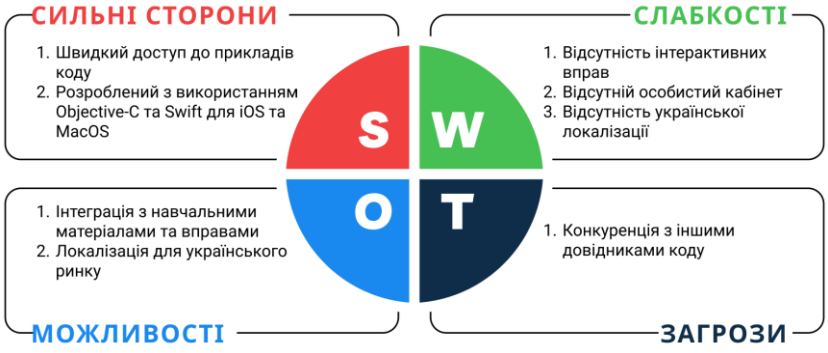


Рис. 1.9. SWOT-аналіз для Java Recipes

Головною перевагою Java Recipes є швидкий доступ до перевірених зразків коду для конкретних задач.

Однак слід визнати, що цей застосунок виступає більше у ролі довідника прикладів, ніж повноцінного навчального інструменту. Йому бракує структурованих навчальних матеріалів, інтерактивних вправ та можливостей для практики, що обмежує його застосування для комплексного навчального процесу.

Настільні застосунки зазвичай пропонують повноцінне середовище розробки, можливість працювати офлайн та багатші функціональні можливості порівняно з веб чи мобільними застосунками. Однак їм часто бракує структурованих навчальних матеріалів, персоналізації та зручної системи відстеження прогресу користувачів.

Підсумовуючи аналіз наявних рішень, можна зробити висновок, що кожна категорія продуктів має свої сильні та слабкі сторони. Вебзастосунки та онлайн-платформи забезпечують зручний доступ та не вимагають встановлення, але часто обмежені функціоналом та відсутністю персоналізації. Мобільні застосунки портативні, але не замінюють повноцінного середовища розробки. Настільні застосунки пропонують більше можливостей, але можуть бракувати структурованості та зручної системи відстеження прогресу.

Результати аналізу виявили відсутність комплексного рішення, яке б поєднувало переваги настільного застосунку, інтерактивного навчального контенту, персоналізації, аналізу прогресу та підтримки української мови на вітчизняному ринку. Ця прогалина підкреслює актуальність та необхідність розробки запропонованого інтерактивного навчального посібника на основі JavaScript фреймворку Electron.

1.3. Обґрунтування потреби у розробці застосунку інтерактивного посібника

Попри на наявність різноманітних навчальних ресурсів для вивчення програмування, як в Інтернеті, так і у вигляді настільних та мобільних застосунків, результати ретельного аналізу наявних рішень виявили відсутність комплексного інтерактивного навчального посібника, який би

повністю задовольняв потреби користувачів, особливо слід звернути увагу, що на українському ринку взагалі не має таких застосунків. Цей факт підкреслює нагальну необхідність розробки нового високоякісного програмного продукту в цій галузі.

Основні причини та потреби, які обґрунтовують розробку інтерактивного посібника на основі JavaScript-фреймворку Electron, можна сформулювати так:

1. Відсутність комплексного рішення для інтерактивного навчання програмування. Попри наявність окремих навчальних ресурсів, жоден з продуктів що існують не пропонує повноцінного поєднання структурованого навчального контенту, інтерактивних вправ, персоналізації та зручного відстеження прогресу в одному зручному застосунку для настільного комп’ютера.

2. Брак локалізованих українськомовних рішень. Більшість наявних продуктів орієнтовані на англомовну аудиторію, що може створювати перепони для користувачів в Україні, які віддають перевагу навчанню рідною мовою або мають недостатній рівень володіння англійською.

3. Обмежений функціонал у веб та мобільних версіях. Вебзастосунки та мобільні застосунки часто обмежені у можливостях та функціоналі порівняно з настільними програмами, особливо в контексті персоналізації та додаткових інструментів.

4. Потреба у зручному офлайн-доступі. Багато користувачів, особливо тих, хто перебуває у місцевостях з нестабільним інтернет-з'єднанням або обмеженим трафіком, віддають перевагу можливості працювати з навчальними матеріалами без необхідності постійного підключення до Інтернету, що забезпечується настільним застосунком.

5. Попит на персоналізацію та відстеження прогресу. Користувачі потребують можливості створювати особисті облікові записи, відстежувати свій прогрес у навчанні, аналізувати допущені помилки, зберігати власні

нотатки та мати доступ до статистики для кращого засвоєння матеріалу та об'єктивної оцінки своїх навичок.

6. Необхідність інтерактивності та практичного досвіду. Ефективне навчання програмування вимагає не лише теоретичних пояснень, але й можливості застосовувати знання на практиці через інтерактивні вправи, завдання різного рівня складності.

7. Наростальний попит на навчання програмування. У сучасному світі, де цифрові технології відіграють все більшу роль, спостерігається інтерес, що росте до вивчення програмування серед різних верств населення, від школярів до професіоналів, які бажають розширити свої навички. Тому існує значний попит на якісні навчальні ресурси, доступні для широкої аудиторії.

8. Поєднання теорії та практики. Оптимальне поєднання теоретичних пояснень, візуалізацій, прикладів коду та практичних завдань дозволить користувачам отримати цілісне розуміння концепцій програмування та набути досвіду їх застосування в реальних ситуаціях.

9. Модульність та масштабованість. Використання JavaScript-фреймворку Electron для розробки посібника забезпечить модульність та масштабованість застосунку, що полегшить додавання нових навчальних модулів, оновлення контенту та розширення функціонала в майбутньому.

10. Сприяння самонавчанню та неперервному професійному розвитку. Інтерактивний посібник, розроблений з урахуванням принципів самонавчання, надасть користувачам можливість самостійно опановувати нові технології та розвивати свої навички програмування, що є критично важливим у швидкозмінному світі інформаційних технологій. Така здатність до безперервного професійного розвитку забезпечить конкурентоспроможність фахівців на ринку праці.

11. Сприяння популяризації програмування. Доступність інтерактивного посібника з українською локалізацією може сприяти популяризації програмування серед українців, заохочувати більше людей до

опанування цієї сфери та розширювати базу потенційних фахівців у галузі інформаційних технологій в Україні.

12. Підвищення якості освіти в IT-галузі. Створення високоякісного навчального ресурсу з урахуванням передових методик та технологій може підвищити загальну якість освіти у сфері інформаційних технологій, забезпечуючи майбутніх фахівців необхідними теоретичними знаннями та практичними навичками для успішної кар'єри у цій галузі.

Розробка інтерактивного посібника на основі JavaScript-фреймворку Electron дозволить задовольнити ці численні потреби, створивши комплексний, локалізований, кросплатформний настільний застосунок зі структурованими навчальними матеріалами, інтерактивними вправами, персоналізацією та зручною системою відстеження прогресу. Такий підхід забезпечить користувачам в Україні доступ до високоякісного навчального ресурсу, що сприятиме ефективному вивченню програмування, розвитку професійних навичок та підготовці кваліфікованих кадрів у галузі інформаційних технологій.

РОЗДІЛ 2. РОЗРОБКА ІНТЕРАКТИВНОГО ЗАСТОСУНКУ

2.1. Вибір технологій та інструментів реалізації

Успішна реалізація будь-якого програмного продукту значною мірою залежить від правильного вибору технологій та інструментів розробки. Ретельно обрані інструменти можуть суттєво підвищити продуктивність роботи, полегшити процес розробки та тестування, забезпечити високу якість кінцевого продукту. З іншого боку, неправильний вибір технологій може призвести до численних проблем, затримок у розробці та, як наслідок, до незадовільного результату. Розглянемо ключові технології та інструменти, які використовувалися для створення інтерактивного посібника на основі JavaScript-фреймворку Electron.

Electron – це фреймворк з відкритим кодом, створений компанією GitHub для розробки кросплатформених застосунків, використовуючи вебтехнології такі як JavaScript, HTML та CSS [8]. Цей фреймворк дозволяє розробникам створювати настільні застосунки, використовуючи ті ж самі технології, що і для розробки вебзастосунків. Electron поєднує в собі переваги веброзробки з можливостями настільних застосунків, забезпечуючи доступ до нативних API операційної системи. Основна ідея Electron полягає у вбудовуванні вебрендерера Chromium та середовища виконання Node.js у настільний застосунок [7].

Переваги використання Electron:

- Кросплатформеність: застосунки, створені за допомогою Electron, можуть працювати на різних операційних системах, таких як Windows, macOS та Linux, без необхідності окремої адаптації коду для кожної платформи.
- Використання вебтехнологій: розробники можуть застосовувати свої знання та досвід у веброзробці для створення настільних застосунків, що значно полегшує процес розробки [20].
- Доступ до нативних API: Electron забезпечує доступ до нативних API операційної системи, що дозволяє створювати застосунки з повноцінною функціональністю, подібною до традиційних настільних програм.

- Відкритий вихідний код: Electron є відкритим проєктом з активною спільнотою розробників, що забезпечує постійну підтримку та вдосконалення платформи.
- Велика екосистема пакетів: завдяки використанню Node.js та npm, розробники мають доступ до величезної кількості готових пакетів та бібліотек, що значно спрощує розробку [6].

Недоліки використання Electron:

- Великий розмір застосунків: оскільки Electron містить повноцінний веббраузер (Chromium) та середовище виконання Node.js, розмір збірок застосунків може бути досить великим, що може вплинути на продуктивність.
- Високе споживання ресурсів: через використання веббраузера та Node.js, застосунки на Electron можуть споживати більше оперативної пам'яті та ресурсів процесора порівняно з традиційними нативними застосунками.
- Безпека: оскільки Electron надає доступ до нативних API, існує потенційний ризик безпеки, якщо код не буде ретельно перевірений та захищений.

Для розробки інтерактивного посібника був обраний фреймворк Electron через його здатність забезпечити кросплатформеність застосунку та можливість використовувати вебтехнології. Ця перевага дозволила значно пришвидшити процес розробки та полегшити підтримку застосунку в майбутньому.

Крім того, Electron надає доступ до нативних API операційних систем, що дозволяє створювати застосунки з повноцінною функціональністю, подібною до традиційних настільних програм [7]. Це було критично важливим для реалізації інтерактивного посібника з багатим функціоналом та зручним користувацьким інтерфейсом.

Попри певні недоліки, такі як великий розмір програм та високе споживання ресурсів, переваги Electron переважили, особливо з огляду на масштаб та складність проєкту інтерактивного посібника. Ретельне

налаштування та оптимізація застосунку дозволили мінімізувати негативний вплив цих недоліків.

Для ефективної розробки сучасних програмних продуктів важливо мати потужне та зручне інтегроване середовище розробки (IDE). У процесі створення інтерактивного посібника на основі Electron було використано IDE PhpStorm від компанії JetBrains. Розглянемо детальніше цей інструмент.

PhpStorm – це кросплатформне інтегроване середовище розробки (IDE) для веброзробників, орієнтоване на мови програмування PHP, HTML, CSS та JavaScript. Воно розроблене компанією JetBrains і є частиною їхньої лінійки інтелектуальних IDE.

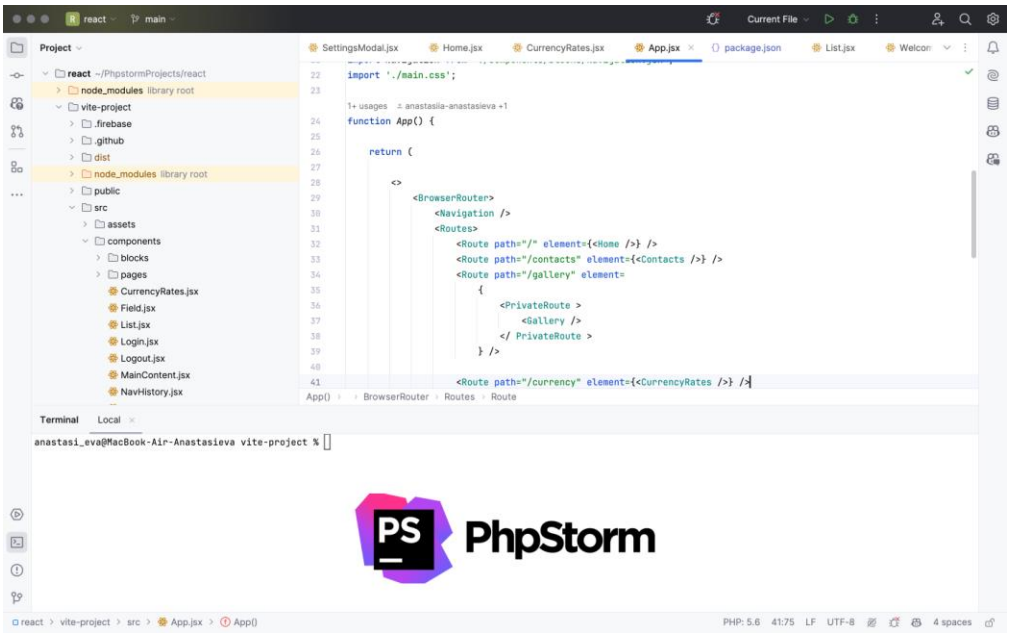


Рис. 2.1. Вигляд IDE PhpStorm

Попри назву, що натякає на орієнтованість на PHP, PhpStorm є потужним інструментом для розробки різноманітних вебзастосунків, включаючи односторінкові застосунки (SPA), мобільні вебзастосунки, прогресивні вебзастосунки (PWA) та застосунки на основі Node.js та Electron.

PhpStorm пропонує низку переваг для розробників, серед яких:

- Інтелектуальний редактор коду: містить підсвічування синтаксису, автодоповнення, підказки для виправлення помилок, рефакторинг коду та багато іншого.

- Інтеграція інструментів: має вбудовані інструменти для налагодження, тестування, інтеграції з системами контролю версій, термінал та інші корисні утиліти.
- Підтримка популярних фреймворків і бібліотек: забезпечує повну підтримку JavaScript-фреймворків (React, Angular, Vue.js), Node.js, Electron та багатьох інших технологій [10].
- Кросплатформність: доступна для Windows, macOS та Linux [15].
- Можливості налаштування: дозволяє налаштувати середовище відповідно до вподобань розробника, включаючи теми, шрифти, комбінації клавіш тощо.

Попри численні переваги, PHPStorm, як і будь-яке інше середовище розробки, має деякі недоліки:

- Комерційна ліцензія: PHPStorm є комерційним продуктом, що вимагає придбання ліцензії. Хоча існує безплатна версія для студентів та викладачів, для комерційного використання потрібно купити ліцензію.
- Високі системні вимоги: PHPStorm є досить ресурсомістким середовищем розробки, тому може вимагати потужного обладнання для забезпечення належної продуктивності.

Переваги PHPStorm, а також можливість використання безоплатної студентської ліцензії, зробили його найкращим вибором для розробки інтерактивного посібника на основі Electron. Завдяки PHPStorm вдалося значно підвищити ефективність роботи над проєктом та створити якісний програмний продукт.

npm (Node Package Manager) – це менеджер пакетів, який використовується для встановлення, оновлення та видалення пакетів (бібліотек) для Node.js. Він є невід'ємною частиною екосистеми Node.js та забезпечує зручний спосіб керування залежностями проєкту.

npm дозволяє легко встановлювати та використовувати сторонні пакети з величезного репозиторію, що містить сотні тисяч відкритих бібліотек для різноманітних цілей: від вебфреймворків до утиліт для розробки. Крім того,

npm забезпечує можливість публікації та спільного використання власних пакетів.

У процесі розробки інтерактивного посібника на основі Electron npm був обраний як система керування пакетами через його тісну інтеграцію з Node.js та Electron. Оскільки Electron заснований на Node.js, використання npm дозволило легко встановлювати та керувати залежностями проєкту, а також забезпечувало доступ до величезної кількості сторонніх бібліотек та утиліт, необхідних для розробки.

Крім того, npm забезпечив централізовану систему оновлень пакетів, що спростило процес підтримки актуальних версій залежностей проєкту та вирішення потенційних конфліктів між ними.

Система контролю версій Git та хостинг GitHub були обрані для розробки застосунку інтерактивного посібника на основі Electron через їх популярність, зручність використання та потужні можливості для роботи над проєктом.

Git забезпечив ефективне відстеження змін у файлах та історії розробки. Використання GitHub надало зручний вебінтерфейс для перегляду коду та управління проєктом, а також забезпечило можливість інтеграції для автоматизації процесів збирання, тестування та розгортання застосунку [14].

Node.js – це кросплатформне середовище виконання JavaScript, яке дозволяє запускати JavaScript-код поза веббраузером [17]. Він широко застосовується для розробки серверних застосунків, API, інструментів для розробки, а також для створення настільних програм у поєднанні з Electron.

Node.js був обраний як платформа для розробки інтерактивного посібника на основі Electron з кількох причин:

- Electron заснований на Node.js, тому використання цієї платформи було необхідним для створення застосунку.
- Асинхронна модель програмування Node.js забезпечує високу продуктивність та масштабованість, що є важливим для створення інтерактивного застосунку [9].

- Кросплатформеність Node.js дозволила розробити застосунок, який може працювати на різних операційних системах без додаткових зусиль.
- Величезна екосистема пакетів npm надала доступ до безлічі корисних бібліотек та утиліт, що значно спростило розробку.
- Використання однієї мови програмування (JavaScript) для клієнтської та серверної частин програми забезпечило єдиний підхід до розробки та полегшило підтримку коду.

У процесі розробки інтерактивного посібника на основі Electron було прийнято рішення використовувати Firebase – це комплексна хмарна платформа для розробки мобільних та вебзастосунків, що розробляється компанією Google [2]. Вона пропонує різноманітні інструменти та сервіси, такі як база даних у реальному часі, хостинг, аутентифікація, аналітика та інші.

Однією з ключових переваг Firebase є можливість швидкого розгортання та масштабування застосунків без необхідності управління серверною інфраструктурою [13]. Крім того, Firebase забезпечує зручну інтеграцію з іншими продуктами Google, такими як Google Analytics та Google Cloud Platform [11].

У процесі розробки інтерактивного посібника Figma була обрана як інструмент для створення дизайну користувацького інтерфейсу та прототипування. Крім того, експорт ресурсів та бібліотеки компонентів Figma зробили процес реалізації дизайну більш швидким та зручним[1].

2.2. Функціональні та нефункціональні вимоги до застосунку

Розробка успішного програмного забезпечення вимагає ретельного планування та чіткого визначення вимог до системи. Вимоги описують функціональність, властивості та обмеження, які повинен задовольняти застосунок для забезпечення належного рівня якості та корисності для користувачів. Тому варто розглянути функціональні та нефункціональні вимоги до інтерактивного посібника на основі JavaScript-фреймворку Electron.

Функціональні вимоги визначають конкретні функції та можливості, які повинен надавати застосунок. Вони описують основні завдання, які має виконувати система, та способи взаємодії користувачів з нею.

У розробці інтерактивного посібника на основі JavaScript-фреймворку Electron функціональні вимоги включають:

1) Загальні вимоги:

а) Реєстрація та автентифікація користувачів: застосунок має дозволяти користувачам створювати облікові записи та проходити автентифікацію для доступу до персоналізованого навчального матеріалу та функцій.

б) Перегляд навчального контенту: користувачі повинні мати змогу переглядати структурований навчальний контент, що включає теоретичну інформацію, приклади коду та практичні завдання для закріплення знань.

в) Проходження тестів та перевірка результатів: застосунок має забезпечувати можливість проходження тестів для оцінки рівня засвоєння матеріалу та отримання зворотного зв'язку щодо результатів.

г) Створення нотаток: користувачі повинні мати можливість створювати та зберігати власні нотатки в межах програми для полегшення процесу навчання.

д) Доступ до особистого кабінету: застосунок має надавати користувачам доступ до персонального кабінету, де вони можуть переглядати своє ім'я, відстежувати прогрес навчання, історію пройдених тестів та аналіз допущених помилок.

2) Вимоги до реєстрації та автентифікації:

а) Реєстрація за допомогою електронної пошти: застосунок повинен підтримувати реєстрацію нових користувачів з використанням їхніх електронних адрес.

б) Безпечне зберігання облікових даних: система має забезпечувати безпечне зберігання та захист облікових даних користувачів,

таких як імена, електронні адреси та паролі, відповідно до найкращих практик безпеки.

3) Вимоги до системи тестування в застосунку:

а) Проходження тестів: застосунок повинен надавати користувачам можливість проходити тести для перевірки рівня засвоєння матеріалу.

б) Аналіз помилок: система має забезпечувати детальний аналіз помилок, допущених користувачем під час проходження тестів, для полегшення процесу навчання та виправлення прогалин у знаннях.

в) Зберігання історії тестів: застосунок повинен зберігати історію пройдених користувачем тестів, надаючи йому можливість відстежувати свій прогрес та повертатися до попередніх результатів.

4) Вимоги до особистого кабінету:

а) Відображення імені користувача: особистий кабінет повинен відображати ім'я, прізвище та електрону пошту поточного користувача.

б) Відображення прогресу навчання: користувачі повинні мати можливість переглядати свій поточний прогрес у вивченні навчального контенту в особистому кабінеті.

в) Доступ до історії тестів та аналізу помилок: в особистому кабінеті користувачі повинні мати доступ до історії пройдених тестів та детального аналізу допущених помилок.

г) Можливість створення нотаток: застосунок має надавати користувачам можливість створювати та зберігати власні нотатки в межах особистого кабінету для полегшення процесу навчання.

Нефункціональні вимоги описують загальні властивості та характеристики інтерактивного посібника, які не пов'язані безпосередньо з його функціональністю, але є критично важливими для забезпечення належної якості та корисності для користувачів.

Нефункціональні вимоги включають:

1) Вимоги до продуктивності:

а) Швидке завантаження та відгук застосунку: інтерактивний посібник повинен забезпечувати швидке завантаження та відгук на дії користувача, щоб забезпечити безперебійний досвід роботи.

2) Вимоги до зручності використання:

а) Інтуїтивно зрозумілий інтерфейс: інтерфейс застосунку повинен бути інтуїтивно зрозумілим та зручним для користувача, забезпечуючи легку навігацію, чітку структуру та зрозумілі елементи керування.

б) Підтримка темних та світлих тем оформлення: застосунок має надавати користувачам можливість вибору між темною та світлою темами оформлення відповідно до їхніх переваг та умов освітлення.

3) Вимоги до сумісності:

а) Кросплатформна сумісність (Windows, macOS, Linux): Інтерактивний посібник, розроблений на основі фреймворку Electron, повинен бути сумісним з основними операційними системами, такими як Windows, macOS та Linux, забезпечуючи однаковий досвід роботи для користувачів на різних платформах.

4) Вимоги до підтримки та оновлення

а) Можливість легкого оновлення навчального контенту: система має бути спроєктована таким чином, щоб полегшити процес оновлення навчального контенту, забезпечуючи можливість вносити зміни та доповнення без значних зусиль та ризику порушення роботи застосунку.

Ці функціональні та нефункціональні вимоги визначають основні очікування та характеристики інтерактивного посібника, що розробляється. Їх ретельне дотримання є ключовим для забезпечення високої якості, корисності та задоволеності користувачів кінцевим продуктом.

2.3. Структура та архітектура застосунку

Одним з ключових аспектів розробки програмного забезпечення є розуміння архітектури та структури застосунку. Це дозволяє ефективно

планувати, реалізовувати та підтримувати програмні рішення. Розглянемо архітектуру та структуру інтерактивного посібника, розробленого на основі JavaScript-фреймворку Electron. Ця інформація допоможе зрозуміти внутрішню організацію проєкту, взаємодію між його компонентами та загальну концепцію роботи застосунку.

Ключовою особливістю Electron є його двопроцесорна архітектура, яка складається з основного процесу (main process) та рендерного процесу (renderer process).

Основний процес відповідає за управління життєвим циклом застосунку, створення вікон, взаємодію з операційною системою та файловою системою. Він також забезпечує доступ до нативних API операційної системи, таких як діалогові вікна, меню та Push-сповіщення.

Рендерний процес, з іншого боку, відповідає за відображення вебконтенту та виконання коду JavaScript, пов'язаного з інтерфейсом користувача. Він працює у ізольованому від основного процесу та інших рендерних процесів середовищі. Кожне вікно застосунку має власний рендерний процес, що дозволяє забезпечити їх незалежність та ізоляцію.

Взаємодія між основним та рендерним процесами відбувається за допомогою асинхронного обміну повідомленнями через механізм IPC (Інтерпроцесна комунікація). Основний процес може надсилати повідомлення до рендерного процесу та навпаки, що дозволяє їм обмінюватися даними та координувати свою роботу.

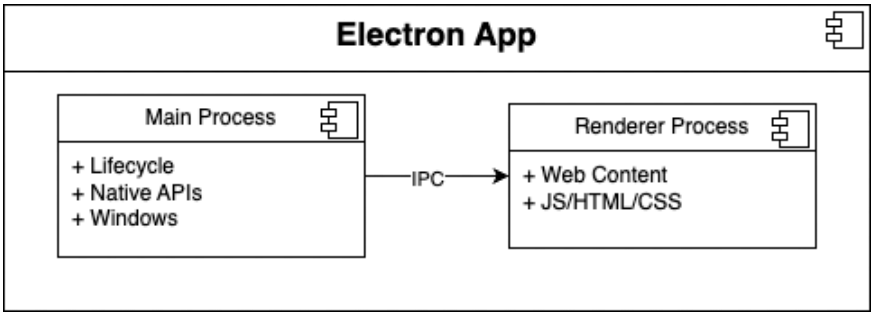


Рис. 2.2. Діаграма компонентів взаємодії між процесами Electron

На діаграмі компонентів зображено основний процес, який відповідає за управління життєвим циклом, взаємодію з нативними API та створення вікон.

Рендерний процес відображає вебконтент, використовуючи HTML, CSS та JavaScript. Обидва процеси взаємодіють за допомогою механізму IPC (Інтерпроцесна комунікація), обмінюючись повідомленнями та даними.

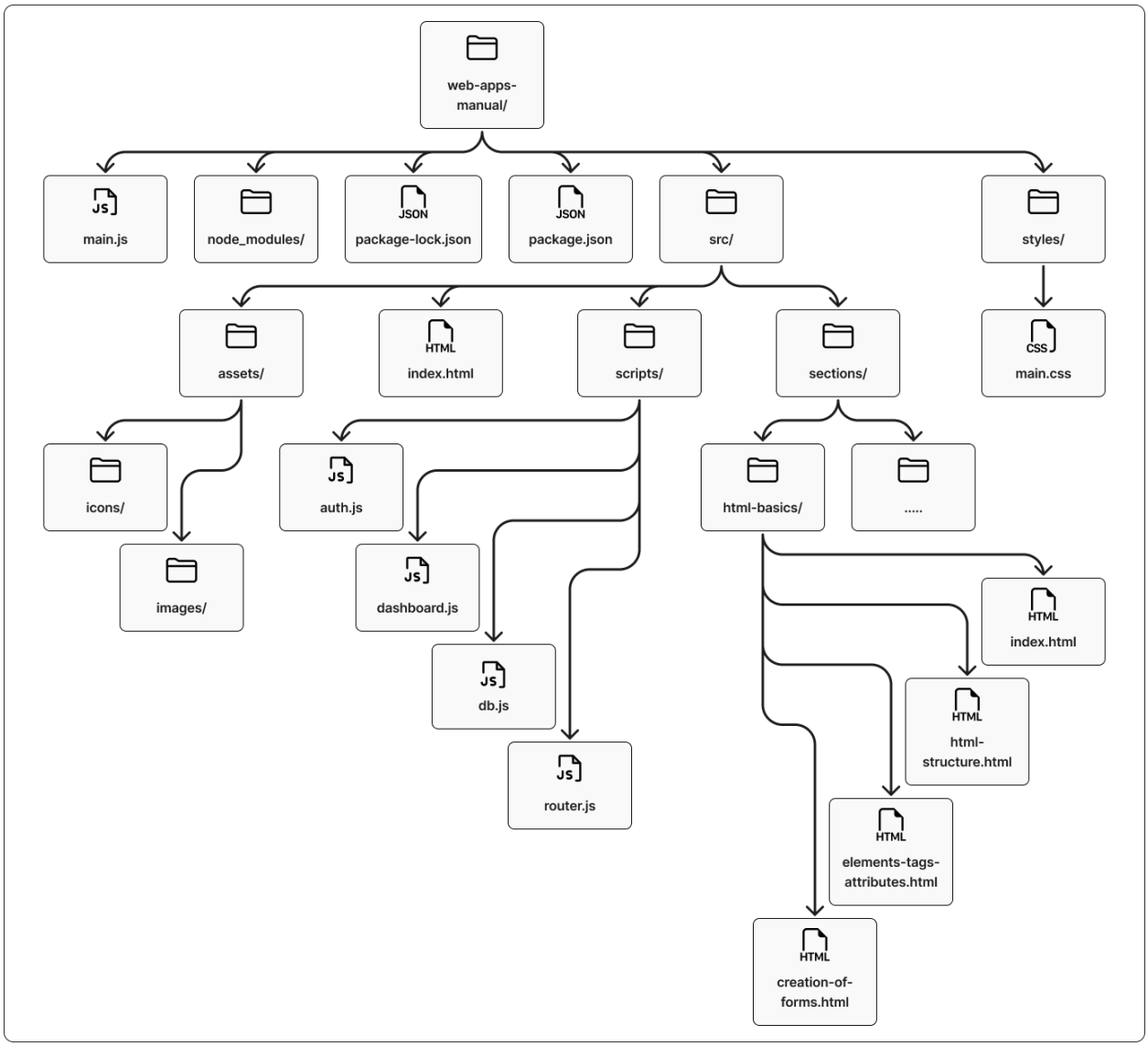


Рис. 2.3. Дерево тек проєкту

Характеристика основних компонентів та тек проєкту:

1) Main.js: це головний файл JavaScript, який є точкою входу для основного процесу Electron. Він відповідає за створення головного вікна застосунку, обробку системних подій та керування життєвим циклом програми.

2) Node_modules: ця тека містить всі встановлені залежності та модулі Node.js, необхідні для роботи проєкту. Вона створюється автоматично при встановленні пакетів за допомогою менеджера пакетів npm або yarn.

3) Package.json та Package-lock.json: це конфігураційні файли Node.js, які містять метадані про проєкт, списки залежностей та інші налаштування. Вони використовуються менеджерами пакетів для встановлення та керування залежностями.

4) Src: ця тека містить весь вихідний код застосунку, включаючи HTML, CSS, JavaScript та додаткові активи, такі як зображення та іконки.

5) Src/Assets: ця тека призначена для зберігання статичних файлів, таких як зображення та іконки, які використовуються в інтерфейсі користувача.

6) Src/Index.html: це головний HTML-файл, який визначає початкову структуру інтерфейсу користувача та завантажує необхідні скрипти та стилі.

7) Src/Scripts/: ця тека містить JavaScript-файли, відповідальні за різну функціональність застосунку, таку як авторизація, взаємодія з базою даних, маршрутизація та інші.

8) Src/Sections/: ця тека організована за розділами посібника і містить HTML-файли для кожного розділу та підрозділу, а також містить HTML для особистого кабінету та авторизації. Кожен розділ має власну теку, яка містить відповідні HTML-файли та, можливо, додаткові ресурси.

9) Styles/Main.css: це головний CSS-файл, який містить стилі для всього застосунку. Він імпортується в HTML-файлах для застосування стилів до інтерфейсу користувача.

Така структура дозволяє легко знаходити та модифікувати різні компоненти застосунку. Вона також полегшує масштабування проєкту.

Інтерактивний посібник на основі Electron має модуль авторизації та реєстрації, який відповідає за забезпечення безпечного доступу користувачів до застосунку та персоналізованого досвіду навчання. Цей модуль дозволяє користувачам створювати облікові записи, автентифікуватися в системі та отримувати доступ до відповідних функцій та контенту.

Опис функціонала модуля:

1) Реєстрація нового користувача: модуль надає форму реєстрації, де користувачі можуть ввести необхідні дані, такі як ім'я, прізвище, електронну пошту та пароль. Після успішної реєстрації, облікові дані користувача зберігаються в базі даних, забезпечуючи їх майбутню автентифікацію.

2) Автентифікація користувача: модуль надає форму входу, де зареєстровані користувачі можуть ввести свої облікові дані (електронну пошту та пароль) для автентифікації в системі. Після успішної автентифікації, користувачам надається доступ до їхнього особистого кабінету та відповідного контенту.

3) Керування сесіями користувачів: модуль відстежує та підтримує сесії користувачів, забезпечуючи безперервний доступ до застосунку після успішної автентифікації. Він також забезпечує безпечний вихід користувачів із системи та скидання їхніх сесій.

4) Інтеграція з Firebase: модуль авторизації та реєстрації інтегрований з Firebase, хмарною платформою для розробки застосунків, яка надає зручні інструменти для автентифікації користувачів та зберігання даних.

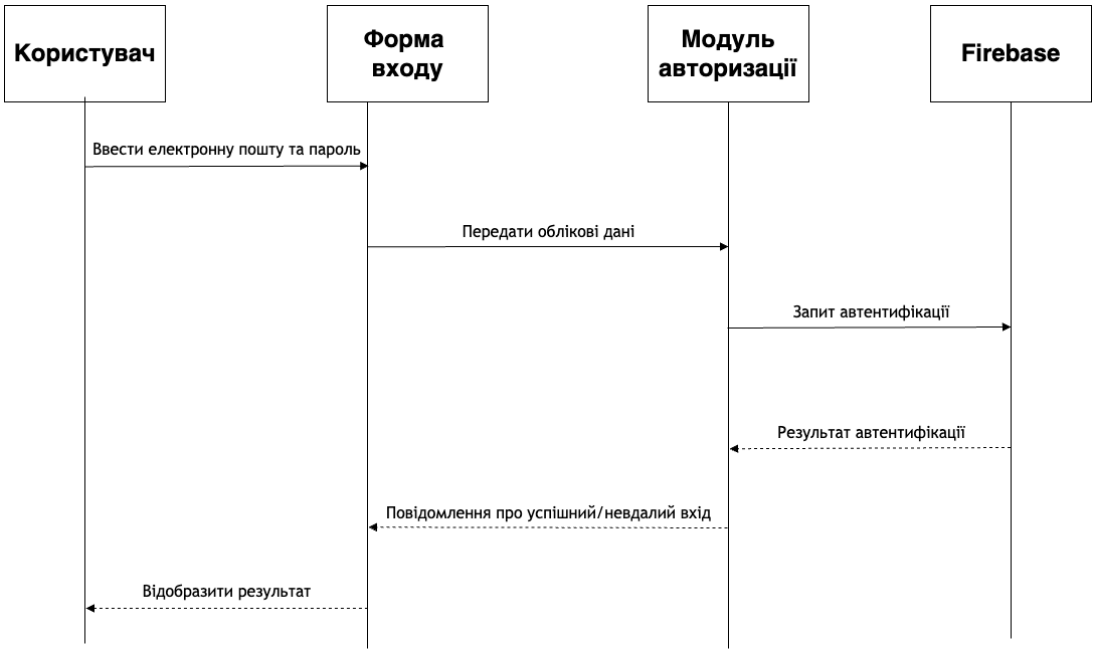


Рис. 2.4. Діаграма послідовності для процесу авторизації

Під час процесу авторизації користувач взаємодіє з формою входу, вводячи свої облікові дані (електронну пошту та пароль). Форма входу передає ці дані модулю авторизації, який, своєю чергою, надсилає запит на

автентифікацію до Firebase. Firebase обробляє запит і повертає результат автентифікації модуля авторизації. Залежно від результату, модуль авторизації повідомляє форму входу про успішний або невдалий вхід, а форма відображає цю інформацію користувачеві.

Рис. 2.5. Діаграма послідовності процесу реєстрації

Процес реєстрації нового користувача починається з введення необхідних даних (ім'я, прізвище, електронна пошта, пароль) у формі реєстрації. Ці дані передаються модулю авторизації, який надсилає запит на створення облікового запису до Firebase. Firebase обробляє запит і повертає результат створення облікового запису модуля авторизації. Залежно від результату, модуль авторизації повідомляє форму реєстрації про успішну або невдалу реєстрацію, а форма відображає цю інформацію користувачеві.

Обидва процеси покладаються на інтеграцію з Firebase, яка забезпечує безпечну автентифікацію користувачів та зберігання їхніх облікових даних. Модуль авторизації виступає посередником між інтерфейсом користувача (формами) та Firebase, обробляючи запити та повідомляючи користувачів про результати.

Особистий кабінет є центральним компонентом застосунку, який надає користувачам персоналізований досвід навчання та доступ до різноманітних функцій і даних. Після успішної автентифікації користувачі потрапляють до

свого особистого кабінету, де вони можуть відстежувати прогрес навчання, проходити тести та керувати своїми даними.

Опис функціонала особистого кабінету:

1) Інформація про користувача: особистий кабінет відображає основну інформацію про користувача, таку як ім'я, прізвище та електронну пошту. Ця інформація отримується з бази даних Firebase після успішної автентифікації.

2) Прогрес навчання: користувачі можуть відстежувати свій загальний прогрес проходження курсу в інтерактивному посібнику. Ця функція дозволяє їм бачити, наскільки ще потрібно опанувати матеріал.

3) Тести та результати: в особистому кабінеті користувачі мають доступ до списку тестів за різними темами. Вони можуть проходити ці тести та переглядати історію своїх спроб та результатів. Також доступний аналіз помилок, допомагаючи користувачам зрозуміти, де вони припустилися помилок і потребують додаткової практики.

4) Управління нотатками: користувачі можуть створювати та зберігати власні нотатки в межах особистого кабінету. Ця функція дозволяє їм робити замітки під час вивчення матеріалу та повертатися до них пізніше.

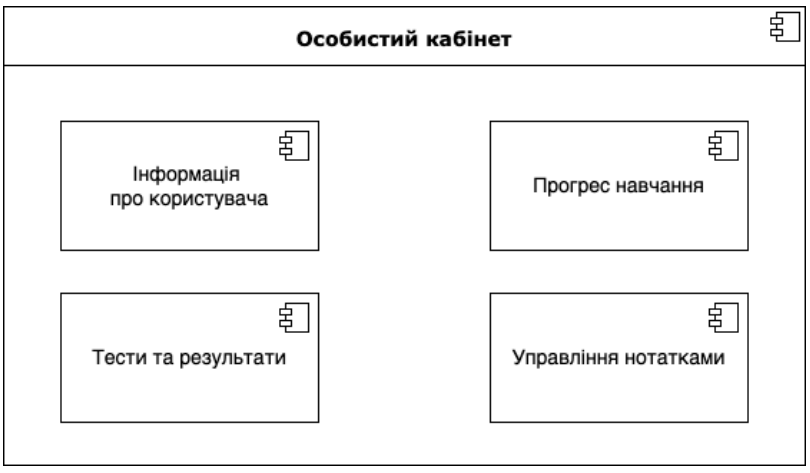


Рис. 2.6. Діаграма компонентів особистого кабінету

На діаграмі компонентів особистого кабінету зображено основні функціональні блоки, такі як відображення інформації про користувача, прогрес навчання, доступ до тестів та результатів, а також управління нотатками. Ці компоненти взаємодіють між собою та з іншими модулями

застосунку, такими як модуль авторизації та база даних Firebase, для забезпечення персоналізованого досвіду користувача.

Одним з ключових аспектів інтерактивного посібника є забезпечення зручної навігації між різними розділами та темами. Для цього застосовується механізм маршрутизації, який дозволяє користувачам переходити від одного розділу до іншого, зберігаючи при цьому стан застосунку та прогрес навчання.

Опис механізму навігації між розділами посібника:

1) Застосунок використовує концепцію односторінкового застосунку (Single Page Application, SPA), де навігація між розділами відбувається без повного перезавантаження сторінки. Замість цього, відповідний контент динамічно завантажується та відображається в межах поточного вікна застосунку.

2) Механізм навігації реалізований за допомогою JavaScript-бібліотеки для маршрутизації, яка дозволяє визначати маршрути для різних розділів та підрозділів посібника. Коли користувач переходить за певним маршрутом (наприклад, шляхом натискання на відповідне посилання в навігаційному меню), бібліотека маршрутизації обробляє цей перехід і завантажує відповідний HTML-файл, що містить контент для цього розділу.

Для забезпечення безпечної авторизації користувачів, зберігання їхніх даних та прогресу навчання, застосунок інтегрований з Firebase. Ця хмарна платформа надає зручні інструменти та сервіси, які полегшують розробку та підтримку застосунків.

Опис використання Firebase для авторизації та збереження даних:

1) Автентифікація користувачів: Firebase Authentication використовується для забезпечення безпечної авторизації користувачів у застосунку. Він пропонує різні методи автентифікації, такі як електронна пошта та пароль, автентифікація через соціальні мережі тощо [12]. Модуль авторизації застосунку взаємодіє з Firebase Authentication для обробки процесів реєстрації, входу та виходу користувачів.

2) Зберігання даних користувачів: для зберігання персональних даних користувачів, таких як ім'я, прізвище, електронна пошта, прогрес навчання та нотатки, застосунок використовує Firebase Cloud Firestore. Цей сервіс надає масштабовані та безпечні рішення для зберігання даних у хмарі, забезпечуючи одночасний доступ та синхронізацію даних між різними клієнтами.

3) Збереження прогресу навчання: прогрес користувача у вивченні матеріалу посібника зберігається в базі даних Firebase. Це дозволяє користувачам припиняти та відновлювати процес навчання в будь-який момент, не втрачаючи досягнутого прогресу. Дані про вивчені розділи, пройдені тести та результати синхронізуються між сеансами користувача.

4) Зберігання нотаток користувачів: застосунок пропонує функцію створення та зберігання нотаток для кожного користувача. Ці нотатки зберігаються в базі даних Firebase.

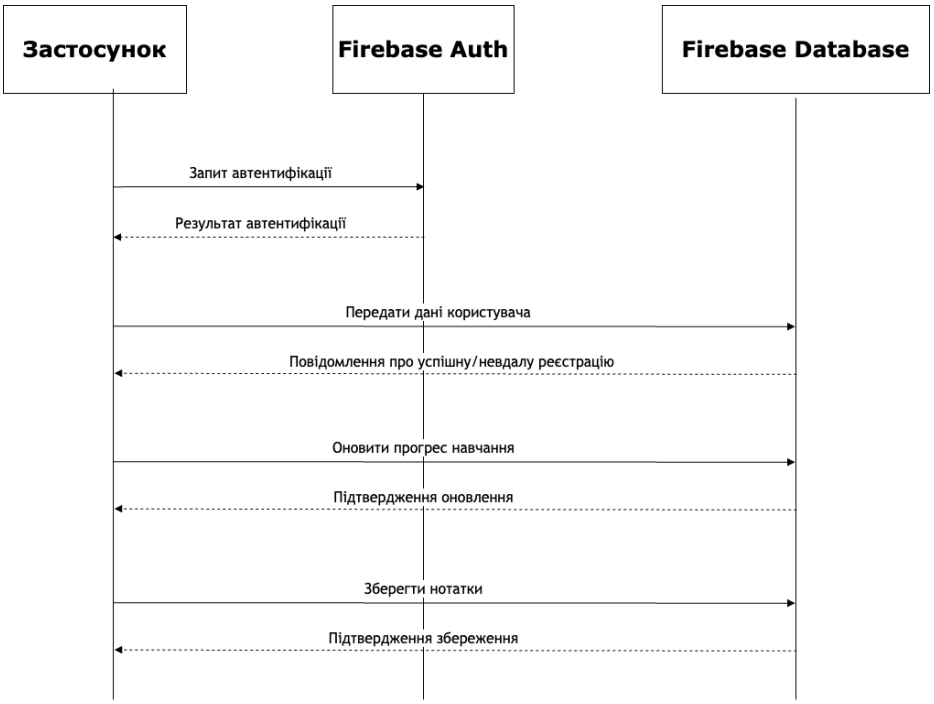


Рис. 2.7. Діаграма послідовності взаємодії з Firebase

Ця діаграма послідовності ілюструє взаємодію між застосунком та різними сервісами Firebase. Застосунок спочатку взаємодіє з Firebase Authentication для автентифікації користувача. Після успішної автентифікації, застосунок зберігає персональні дані користувача, оновлює прогрес навчання

та зберігає нотатки в Cloud Firestore. Firebase підтверджує успішне збереження даних.

Отже, інтерактивний посібник на основі JavaScript-фреймворку Electron має чітко структуровану та ефективну архітектуру з двопроцесорною моделлю Electron, яка забезпечує розподіл обов'язків, ізоляцію між компонентами та підвищену безпеку застосунку.

2.4. Проєктування інтерфейсу користувача

Інтерфейс користувача (UI) є критично важливим компонентом для створення успішного застосунку. Зручний та інтуїтивний інтерфейс забезпечує позитивний досвід користувачів під час взаємодії з застосунком, що, своєю чергою, сприяє ефективному засвоєнню матеріалу. Тому на етапі проєктування особлива увага була приділена розробці зручного та привабливого дизайну інтерфейсу.

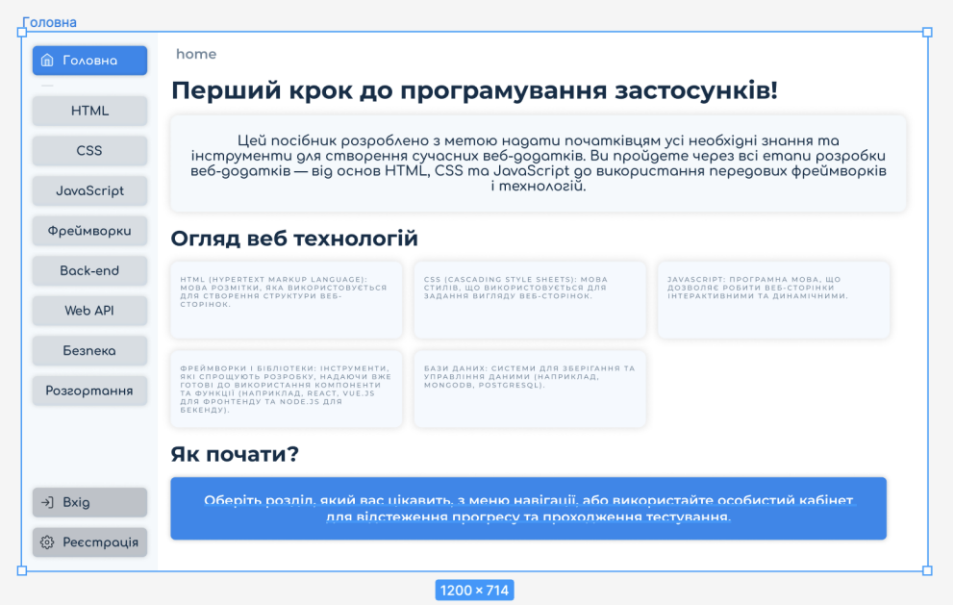


Рис. 2.8. Приклад як виглядає дизайн застосунку в Figma

Загальна концепція дизайну базувалася на принципах юзабіліті (usability) та прагненні створити чистий, мінімалістичний інтерфейс без зайвих елементів, що могли б відвертати увагу користувача від основного навчального контенту. Однак водночас, дизайн мав залишатися сучасним та естетично привабливим.

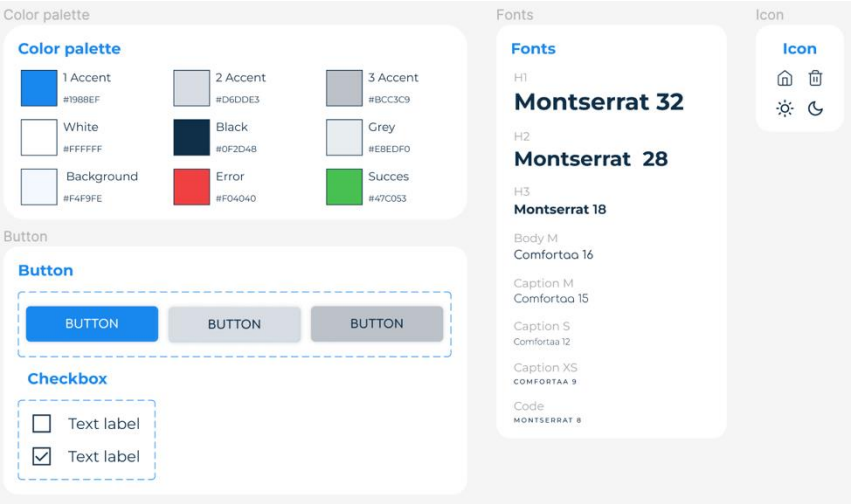


Рис. 2.9. UI kit дизайну застосунку

Для забезпечення послідовності та узгодженості в усьому застосунку була розроблена єдина колірна палітра та набір шрифтів (див. рис. 2.9). Основна колірна гама складалася з білого фону, синього акцентного кольору для ключових елементів інтерфейсу, таких як кнопки та посилання, а також сірого кольору для допоміжних елементів, наприклад, полів вводу.

Додатковими кольорами були чорний для тексту, червоний для позначення помилок та зелений для успішних дій. Типографіка базувалася на гарнітурі Montserrat та Comfortaa з різними розмірами для заголовків, основного тексту, міток та коду.

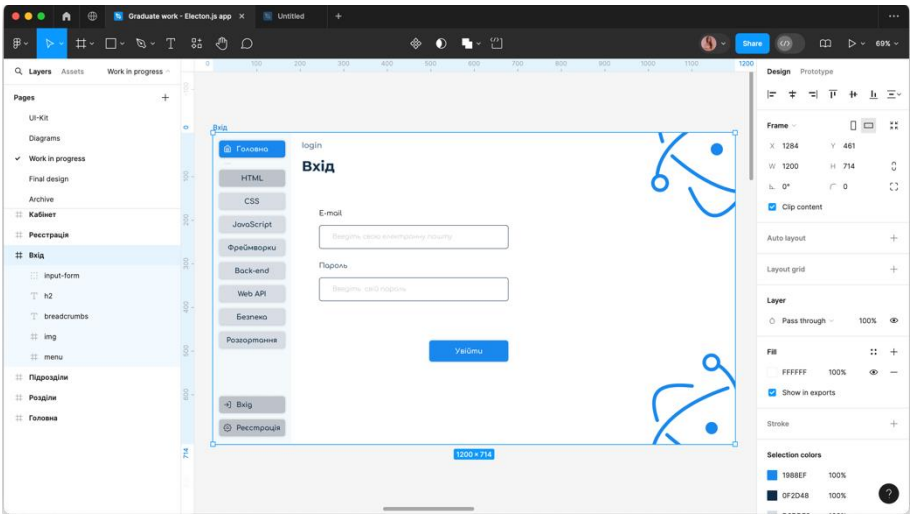


Рис. 2.10. Дизайн екрана входу застосунку в Figma

Починаючи з екрана входу (Login) (див. рис. 2.10), користувач має можливість перейти до розділу реєстрації або відразу увійти до системи, ввівши свої облікові дані.

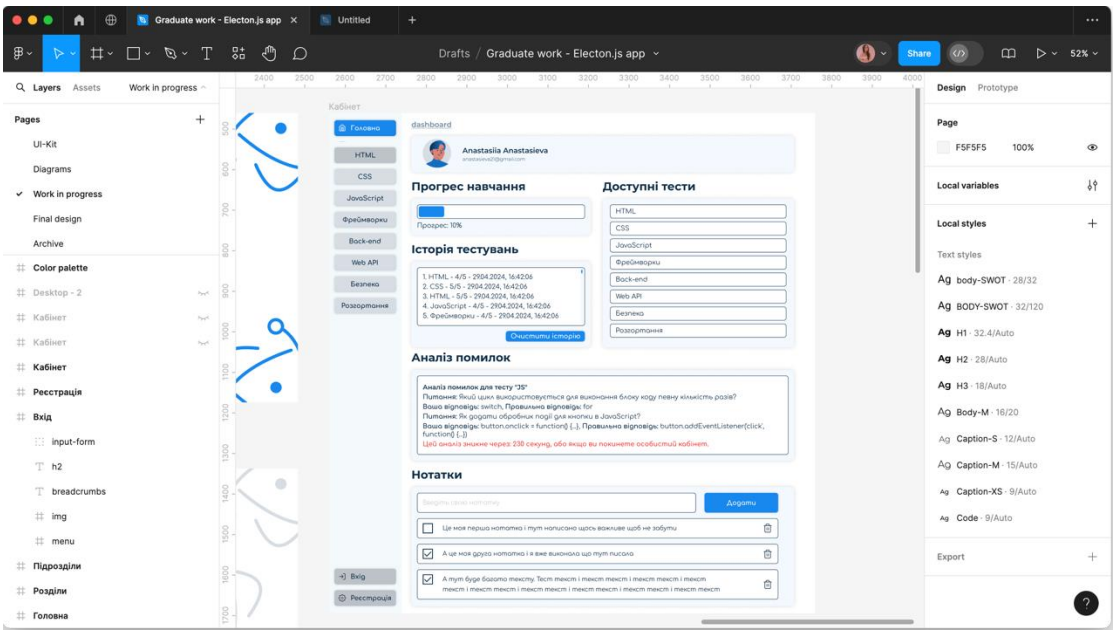


Рис. 2.11. Дизайн екрана особистого кабінету застосунку в Figma

В особистому кабінеті (Dashboard) (див. Рис. 2.11) користувач має доступ до всіх основних функцій та матеріалів. Тут відображалися ім'я користувача, прогрес проходження курсу, історія пройдених тестів, список доступних тестів за розділами, аналіз типових помилок з підказками щодо їх виправлення, а також можливість створювати нотатки безпосередньо в застосунку. Для зручності навігації використовувалася бічна панель з розділами матеріалу.

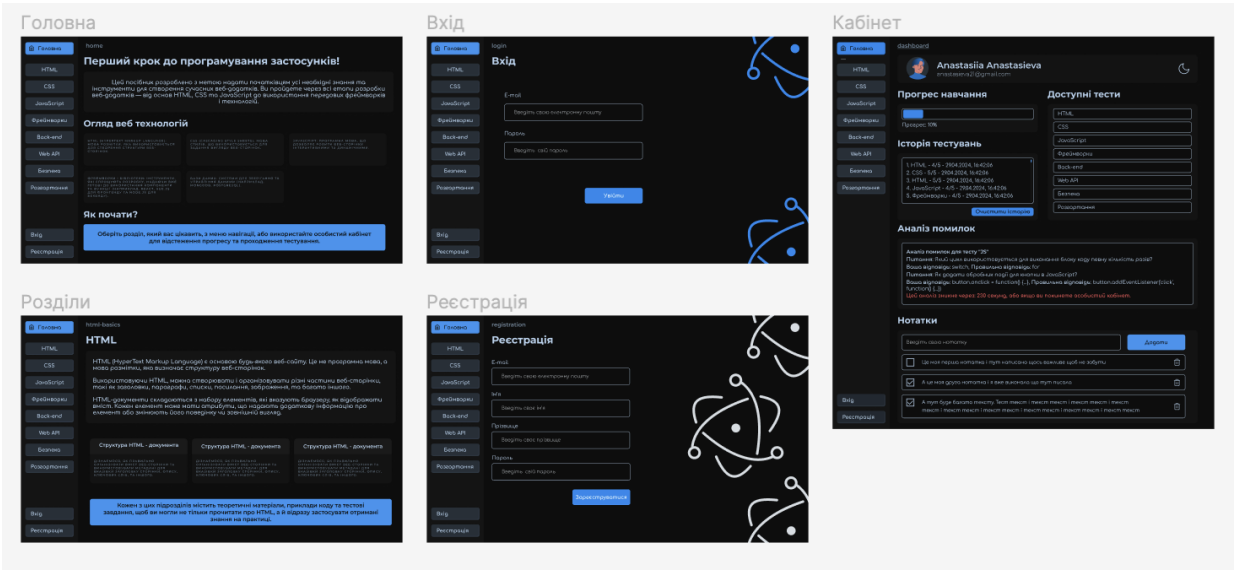


Рис. 2.12. Дизайн екранів з темною темою застосунку в Figma

Окремо варто відзначити темну тему для застосунку (див. рис. 2.12), яка була передбачена для зручності користувачів, які працюють у темних умовах

або віддають перевагу темним інтерфейсам з метою зменшення навантаження на зір.

Такий ретельно спроєктований інтерфейс користувача забезпечив зручність використання інтерактивного посібника, допомагаючи користувачам зосередитися на навчальному матеріалі та ефективно засвоювати нові знання та навички.

2.5. Інтеграція бази даних Firebase

Firebase є потужною хмарною платформою для розробки вебзастосунків та мобільних застосунків, яка надає всі необхідні інструменти та сервіси для зберігання та обробки даних, автентифікації користувачів, хостингу контенту та багато іншого. В цьому випадку Firebase використовувався для впровадження функцій авторизації та збереження даних користувачів.

Для інтеграції Firebase в цей проєкт були виконані наступні кроки:

а) Створення нового проєкту в Firebase Console.

б) Реєстрація вебзастосунка та отримання конфігураційних параметрів (API ключ, ідентифікатор проєкту тощо).

в) Встановлення та імпорт Firebase SDK для JavaScript у вихідний код проєкту.

г) Ініціалізація Firebase SDK з отриманими конфігураційними параметрами.

Для забезпечення функцій авторизації та реєстрації користувачів було використано сервіс Authentication Firebase. Цей сервіс дозволяє користувачам авторизуватися за допомогою електронної пошти та паролю, а також підтримує інші провайдери автентифікації, такі як Google, Facebook, GitHub тощо.

Процес авторизації та реєстрації був реалізований наступним чином:

а) На сторінці входу користувачі можуть ввести свої облікові дані (електронну пошту та пароль) та ініціювати процес автентифікації.

б) На сторінці реєстрації користувачі можуть створити новий обліковий запис, вказавши необхідні дані (ім'я, прізвище, електронну пошту та пароль).

в) Після успішної автентифікації користувач отримує доступ до основної функціональності застосунку, включаючи особистий кабінет та навчальний контент.

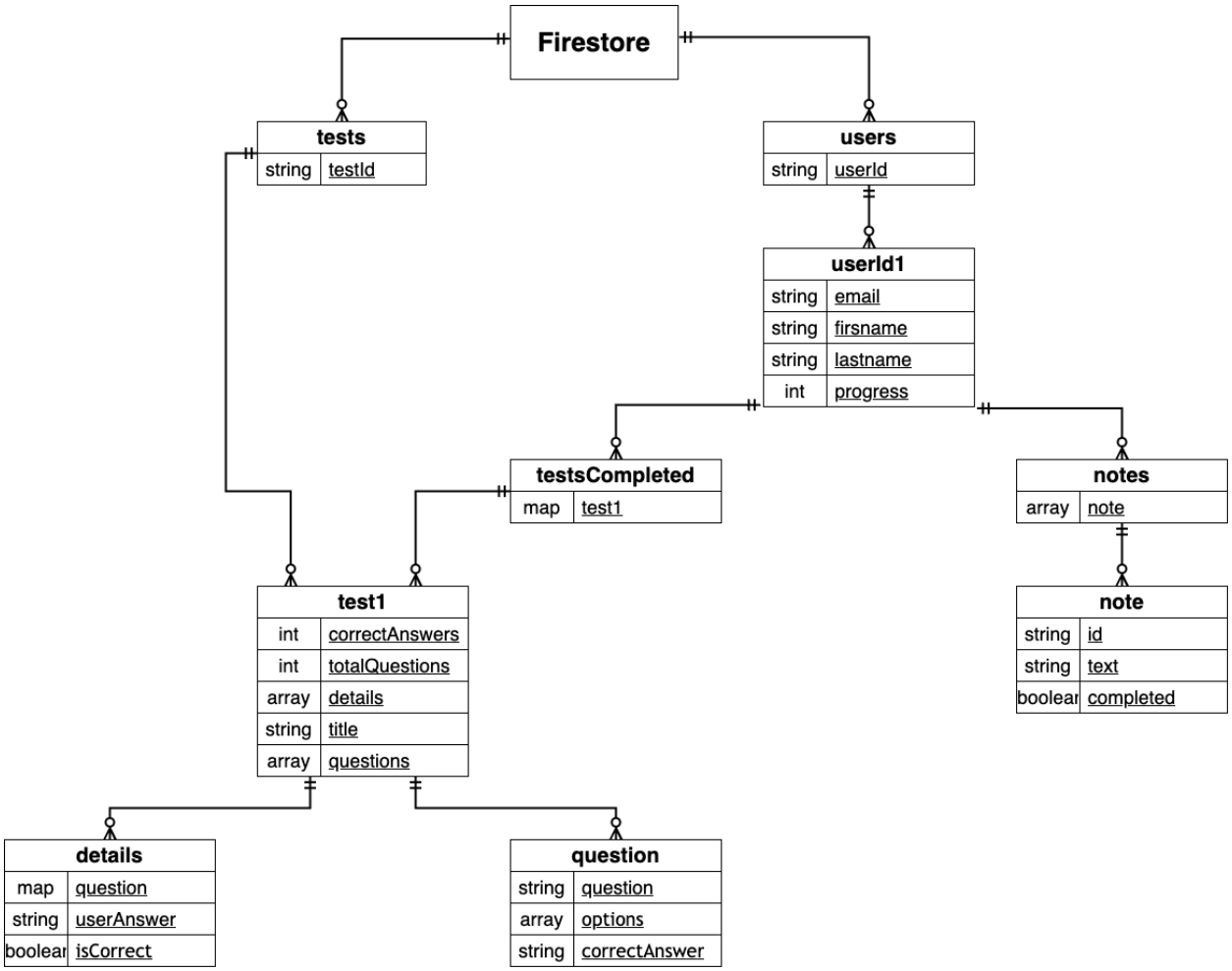


Рис. 2.13. Діаграма бази даних у Firestore для застосунку

Ця діаграма відображає основні сутності та їхні відношення в базі даних Firestore.

1) users - Колекція, що зберігає дані про користувачів.

а) Кожен документ у цій колекції представляє одного користувача і містить поля, такі як email, firstName, lastName, progress, testsCompleted та notes.

б) testsCompleted є мапою, де ключами є ідентифікатори тестів, а значеннями - об'єкти з інформацією про результати тесту, включаючи кількість правильних відповідей, загальну кількість питань та деталі відповідей на кожне питання.

в) notes є масивом об'єктів, що представляють нотатки користувача, з полями id, text та completed.

2) tests - Колекція, що зберігає дані про тести.

а) Кожен документ у цій колекції представляє один тест і містить поля title та questions.

б) questions є масивом об'єктів, кожен з яких містить текст питання, варіанти відповідей та правильну відповідь.

Така структура дозволяє ефективно зберігати та отримувати дані про користувачів, їхні результати тестів та нотатки, а також дані про самі тести та їхні питання.

Зв'язки між сутностями:

1) FirestoreRoot та users: FirestoreRoot містить багато користувачів (users).

2) users та userId1: кожен користувач може мати багато документів userId1.

3) userId1 та testsCompleted: кожен документ користувача може містити багато завершених тестів.

4) testsCompleted та test1: кожен завершений тест може містити багато деталей тесту (test1).

5) test1 та details: кожен тест може містити багато деталей питання (details).

6) userId1 та notes: кожен документ користувача може містити багато нотаток (notes).

7) notes та note: кожна нотатка може містити багато деталей нотатки (note).

8) FirestoreRoot та tests: FirestoreRoot містить багато тестів (tests).

9) tests та test1: кожен тест може містити багато питань (test1).

10) test1 та question: кожен тест містить багато питань (question).

Для взаємодії з Firestore було створено окремий модуль (db.js), який інкапсулює логіку зберігання та отримання даних. Цей модуль експортує функції для збереження та оновлення даних користувача, таких як прогрес навчання, нотатки та результати тестів.

Для забезпечення безпеки та контролю доступу до даних у Firestore були налаштовані відповідні правила безпеки. Ці правила визначають, хто має право читати, записувати або оновлювати дані в базі даних.

```
rules_version = '2';

service cloud.firestore {
  match /databases/{database}/documents {
    match /users/{userId} {
      allow read, write: if request.auth != null && request.auth.uid == userId;
    }
  }
}
```

Рис. 2.14. Код налаштування правил безпеки Firestore

Ці правила дозволяють користувачам читати та записувати дані лише у свій власний документ у колекції users, використовуючи їхній унікальний ідентифікатор користувача (userId), отриманий під час автентифікації. Це запобігає несанкціонованому доступу до даних інших користувачів.

Для тестового періоду, коли автентифікація не була повністю налаштована, було застосовано тимчасове правило, що дозволяло читати та записувати дані без автентифікації. Однак, перед публікацією застосунку, це правило було вилучено для забезпечення належного рівня безпеки.

Загалом, інтеграція Firebase у цей проєкт забезпечила надійне та масштабоване рішення для зберігання даних користувачів, автентифікації та контролю доступу, дозволяючи зосередитися на розробці основної функціональності застосунку.

2.6. Розробка компонентів та функціональності застосунків

Створення якісного інтерактивного посібника на основі JavaScript-фреймворку Electron вимагає ретельної розробки ключових компонентів та

функціональності застосунку. Ці компоненти забезпечують користувачам зручний і ефективний досвід роботи з посібником, дозволяючи їм легко переміщатися між розділами, проходити авторизацію, відстежувати свій прогрес, проходити тестування та персоналізувати інтерфейс згідно з власними уподобаннями.

Кожен з компонентів відіграє важливу роль у забезпеченні зручного та інтуїтивно зрозумілого інтерфейсу, що допомагає користувачам ефективно засвоювати навчальний матеріал. Крім того, розглянемо інтеграцію з сервісами зовнішніх постачальників, такими як Firebase, для зберігання та управління даними користувачів, їх прогресом і налаштуваннями.

Створення головного вікна застосунку є критично важливою частиною у розробці інтерактивного посібника на основі Electron. Воно є першим, що побачить користувач при запуску програми, тому його налаштування повинні бути ретельно продумані для забезпечення належного користувацького досвіду [4].

Головне вікно створюється за допомогою модуля BrowserWindow з основної бібліотеки Electron. У файлі main.js визначаємо функцію createWindow, яка відповідає за створення та налаштування вікна. Ця функція приймає об'єкт опцій, що дозволяє налаштувати різні аспекти вікна, такі як розміри, іконка, заголовок та параметри вебпереглядача.

Наприклад, у наведеному нижче коді створюємо вікно з шириною 1200 пікселів і висотою 714 пікселів, встановлюємо власну іконку та включаємо інтеграцію Node.js та відключення ізоляції контексту у вебпереглядачі:

```
// Функція для створення основного вікна браузера
1+ usages  anastasiia-anastasieva *
function createWindow() : void  {
    // Створення вікна з певними розмірами та вебналаштуваннями
    const mainWindow : BrowserWindow  = new BrowserWindow( options: {
        width: 1200, // Ширина вікна у пікселях
        height: 714, // Висота вікна у пікселях
        icon: path.join(__dirname, 'iconspr/icon'), // Іконка вікна
        webPreferences: {
            nodeIntegration: true, // Дозволяємо використання Node.js
            contextIsolation: false // Відключаємо ізоляцію контексту для використання require
        }
    });

    // Завантаження index.html у вікно.
    mainWindow.loadFile(path.join(__dirname, 'src/index.html'));
```

Рис. 2.15. Код що створює головне вікно застосунку

Налаштування webPreferences є особливо важливими для застосунків на основі Electron, оскільки вони дозволяють використовувати модулі Node.js безпосередньо в контексті вікна. У цьому випадку включаємо nodeIntegration, що дозволяє використовувати модулі Node.js, та відключаємо contextIsolation, що знімає обмеження на використання глобальних об'єктів, таких як require.

Після створення вікна з необхідними налаштуваннями завантажуємо початкову HTML-сторінку за допомогою методу loadFile. Ця сторінка зазвичай містить основну розмітку та навігаційне меню, що дозволяє користувачам переміщатися між різними розділами інтерактивного посібника.

Реалізація навігації та маршрутизації здійснюється у JavaScript-файлі router.js. Центральною функцією в цьому файлі є loadSection, яка відповідає за завантаження вмісту відповідного розділу або підрозділу.

Функція loadSection приймає два аргументи: sectionName (назва розділу) та isSubsection (прапорець, що вказує, чи є це підрозділом). Залежно від цих параметрів, функція визначає відповідний шлях до HTML-файлу, що містить контент розділу, і завантажує його за допомогою fetch.

Наприклад, якщо sectionName дорівнює 'home', функція завантажить файл sections/home.html. Якщо sectionName є назвою підрозділу, наприклад 'html-basics/elements-tags-attributes', функція завантажить файл sections/html-basics/elements-tags-attributes.html.

Після успішного отримання HTML-вмісту функція loadSection вставляє його в елемент <main> на сторінці за допомогою властивості innerHTML. Таким чином, користувач бачить актуальний контент відповідного розділу або підрозділу.

Рис. 2.16. Код що завантажує вміст певного розділу

Крім завантаження вмісту, система навігації також включає інші важливі функціональності, такі як оновлення активного посилання в навігаційному меню та обробка подій для переходу до підрозділів.

Функція updateActiveLink відповідає за додавання класу active до відповідного посилання в навігаційному меню, вказуючи на поточний розділ. Це допомагає користувачеві легше орієнтуватися в структурі посібника.

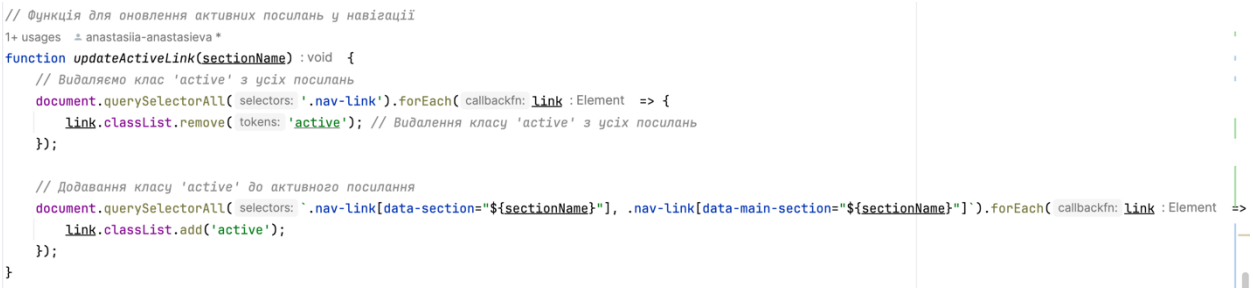


Рис. 2.17. Код функції «updateActiveLink»

Функція attachCardListeners додає обробників подій до карток розділів, що дозволяє користувачам переходити до підрозділів, натискаючи на відповідну картку. При кліку на картку функція визначає, чи є цей розділ

підрозділом, і відповідним чином завантажує його вміст за допомогою функції loadSection.

```
// Функція для навішування обробників подій на картки
1+ usages   anastasiia-anastasieva *
function attachCardListeners() : void  {
    document.querySelectorAll( selectors: '.card').forEach( callbackfn: card : Element  => {
        // При кліку на картку
        card.addEventListener( type: 'click',  listener: () : void  => {
            const section : string  = card.getAttribute( qualifiedName: 'data-section'); // Отримання секції з атрибуту картки
            const mainSection : string  = card.getAttribute( qualifiedName: 'data-main-section'); // Отримання основної секції з атрибуту картки
            const isSubsection : boolean  = section !== mainSection; // Перевірка, чи є поточна секція підрозділом
            loadSection( sectionName: `${mainSection}/${section}`, isSubsection); // Завантаження секції
        });
    });
}
```

Рис. 2.18. Код функції «attachCardListeners»

Функція loadSection отримує два параметри: шлях до розділу та прапорець, який визначає, чи є цей розділ підрозділом. На основі цих параметрів функція завантажує відповідний контент. Якщо розділ є підрозділом, loadSection забезпечує правильне відображення його вмісту, враховуючи загальну структуру та ієрархію застосунку.

Цей підхід забезпечує гнучкість та зручність у навігації між розділами та підрозділами інтерактивного посібника, дозволяючи користувачам швидко переходити до потрібних тем і матеріалів.

Для реалізації авторизації та реєстрації використовуються сервіси Firebase Authentication та Firebase Firestore. Firebase Authentication відповідає за процеси реєстрації нових користувачів, входу в систему та керування обліковими записами, тоді як Firebase Firestore використовується для зберігання та управління даними користувачів.

Логіка для обробки форм входу та реєстрації міститься у файлі auth.js. Цей файл включає обробників подій для відповідних форм, де перевіряються введені дані користувача та виконуються необхідні операції з Firebase Authentication.

Наприклад, при спробі реєстрації нового користувача викликається метод createUserWithEmailAndPassword з Firebase Authentication, передаючи введені користувачем електронну пошту та пароль. У разі успішної реєстрації створюється документ користувача в колекції users у Firebase Firestore за допомогою функції createUserDocument з файлу db.js.

```
// Викликаємо Firebase API для створення нового користувача
firebase.auth().createUserWithEmailAndPassword(email, password)
    .then((userCredential) : void  => {
        console.log("Реєстрація успішна:", userCredential.user);
        // Додаємо збереження імені та прізвища
        createUserDocument(userCredential.user.uid, email, firstName, lastName);
        window.loadSection( sectionName: 'dashboard');
    })
    .catch((error) : void  => {
        console.error("Помилка реєстрації:", error.code, error.message);
        document.getElementById( elementId: 'registerError').textContent = error.message;
    });
```

Рис. 2.19. Код методу «createUserWithEmailAndPassword»

Функція createUserDocument перевіряє, чи існує вже документ користувача в колекції users. Якщо документ не існує, функція створює його з початковими даними, такими як електронна пошта, прогрес навчання, історія проходження тестів та нотатки.

```
// Функція для створення або перевірки існування документа користувача в колекції 'users'
no usages  new *
function createUserDocument(user, callback) : void  {
    const userRef : DocumentReference<DocumentData>  = db.collection( collectionPath: 'users').doc(user.uid);

    // Спроба отримати документ
    userRef.get().then(doc : DocumentSnapshot<DocumentData>  => {
        if (!doc.exists) {
            // Якщо документ не існує, створюємо його з початковими даними
            userRef.set({
                email: user.email, // Зберігаємо email користувача
                progress: 0, // Зберігаємо початковий прогрес
                testsCompleted: [], // Масив завершених тестів
                notes: [] // Масив нотаток
            }, {merge: true})
                .then(() => callback(null)) // Виклик callback-функції без помилок
                .catch((error) => callback(error)); // Обробка можливих помилок при збереженні
        } else {
            // Якщо документ вже існує, повертаємо успіх без створення нового
            callback(null);
        }
    }).catch((error) : void  => {
        console.error("Error checking user document:", error); // Логування помилок
        callback(error);
    });
}
```

Рис. 2.20. Код функції «createUserDocument»

Аналогічним чином реалізовано процес входу в систему, де використовується метод signInWithEmailAndPassword з Firebase Authentication для перевірки введених користувачем облікових даних.

Крім того, також реалізовується обробник події для виходу з системи, який викликає метод signOut з Firebase Authentication та завантажує головну сторінку посібника.

Завдяки інтеграції з Firebase Authentication та Firebase Firestore забезпечується безпечна авторизація та реєстрація користувачів, а також зберігання їх персональних даних для подальшого використання в інших компонентах застосунку, таких як профіль користувача.

Після успішної авторизації або реєстрації користувач потрапляє на свою персональну панель, яка є одним з ключових компонентів інтерактивного посібника. Ця панель надає користувачеві доступ до різноманітних функцій та персоналізованої інформації, що допомагає ефективно відстежувати прогрес навчання та керувати своїми даними.

Розмітка профілю користувача знаходиться у файлі sections/dashboard/index.html. Цей файл містить HTML-структуру для відображення інформації про користувача, прогресу навчання, доступних тестів, історії проходження тестів, аналізу помилок та нотаток.

Одним з ключових елементів в профілі користувача є секція "Прогрес навчання". Ця секція відображає поточний прогрес користувача у вивченні матеріалу у вигляді прогрес-бару та відсоткового значення. Дані для відображення прогресу зчитуються з документа користувача у Firebase Firestore і оновлюються динамічно при проходженні тестів або вивченні нових розділів.

```
<div id="learning-progress">
    <div class="progress-container">
        <div class="progress-bar" id="progress-bar"></div>
    </div>
    <p>Прогрес: <span id="progress-percentage">0</span>%</p>
</div>
```

Рис. 2.21. Код секції «Прогрес навчання»

Крім того, на панелі користувача відображається список доступних тестів, які користувач може проходити для перевірки своїх знань. Ці тести завантажуються з певного джерела даних (наприклад, в такому випадку з бази даних) і відображаються у вигляді списку.

```
<div id="available-tests">
    <ul id="tests-list">
        <!-- Тести будуть тут -->
    </ul>
</div>
```

Рис. 2.22. Код секції «Доступні тести»

Історія проходження тестів також відображається на панелі користувача. Ця інформація зберігається в документі користувача у Firebase Firestore і завантажується для відображення у вигляді списку. Користувач може переглядати дати та результати своїх попередніх спроб.

```
    <!-- Історія тестувань -->
    <h3>Історія тестувань</h3>
    <div id="test-history">
        <ul id="history-list">
            <!-- Записи історії будуть тут -->
        </ul>
        <button id="clear-history-button">Очистити історію</button>
    </div>
</div>
```

Рис. 2.23. Код секції «Історія тестів»

Іншим важливим компонентом на панелі користувача є секція "Аналіз помилок". Ця секція відображає детальну інформацію про помилки, допущені користувачем під час проходження тестів. Аналіз помилок допомагає користувачеві зрозуміти, в яких областях він потребує додаткового вивчення матеріалу.

```
<div id="error-analysis">
    <h3>Аналіз помилок</h3>
    <div class="fon-analysis">
        <div class="container-analysis">
            <div id="analysis-content"></div>

            <div id="error-analysis-timer"></div>
        </div>
    </div>
</div>
```

Рис. 2.24. Код секції «Аналіз помилок»

На панелі користувача є секція для ведення нотаток. Користувач може додавати нові нотатки за допомогою текстового поля та кнопки "Додати". Ці нотатки зберігаються в документі користувача у Firebase Firestore й

автоматично завантажуються та відображаються у вигляді списку при відкритті панелі користувача.

```
<h3>Нотатки</h3>
<div id="todo-list">
    <label for="new-task"></label><input type="text" id="new-task" placeholder="Введіть свою нотатку"/>
    <button id="add-task-button">Додати</button>
    <ul id="tasks"></ul>
</div>
```

Рис. 2.25. Код секції для введення нотаток

Логіка для роботи з нотатками міститься у файлі db.js. Функція addTaskForUser дозволяє додавати нову нотатку для певного користувача, оновлюючи масив notes у його документі в Firestore.

```
// Функція для додавання нової задачі користувачеві
1+ usages  new *
function addTaskForUser(userId, taskText, callback) : void  {
    const userRef : DocumentReference<DocumentData>  = db.collection( collectionPath: 'users').doc(userId); // Отримання посилання на документ користувача
    const newTask : {completed: boolean, text: any}  = {text: taskText, completed: false}; // Створення нової задачі

    // Оновлення документа з додаванням нової задачі
    userRef.update( data: {
        notes: firebase.firestore.FieldValue.arrayUnion(newTask) // Додавання задачі до масиву 'notes'
    })
        .then(() => callback(null)) // Успішне виконання без помилок
        .catch((error) => callback(error)); // Обробка помилок
}
```

Рис. 2.26. Код функції «addTaskForUser»

Функція getTasksForUser завантажує всі наявні нотатки користувача з його документа в Firestore і передає їх у callback-функцію для подальшого відображення.

```
// Функція для отримання всіх задач користувача
1+ usages  new *
function getTasksForUser(userId, callback) : void  {
    db.collection( collectionPath: 'users').doc(userId).get() Promise<DocumentSnapshot<...>>
        .then(doc : DocumentSnapshot<DocumentData>  => {
            if (doc.exists) {
                callback(null, doc.data().notes); // Повернення задач, якщо документ існує
            } else {
                callback(new Error("No such document!"), null); // Повідомлення про відсутність документу
            }
        }) Promise<void>
        .catch((error) : void  => {
            callback(error, null); // Обробка помилок під час запиту
        });
}
```

Рис. 2.27. Код функції «getTasksForUser»

Крім секції для нотаток, на панелі користувача також присутня функціональність для перемикання між світлою та темною темами

оформлення. Ця функціональність реалізована у функції setupThemeToggle у файлі db.js. При натисканні на кнопку перемикача теми до елемента <body> додається або видаляється клас dark-theme, що змінює стилі оформлення відповідно до обраної теми. Поточна тема зберігається у localStorage браузера для збереження налаштувань між сеансами.

Таким чином, панель користувача забезпечує зручний та інформативний інтерфейс, надаючи користувачам доступ до їх персональних даних, прогресу навчання, тестів, аналізу помилок та можливість ведення нотаток. Ця функціональність є ключовою для забезпечення персоналізованого та ефективного досвіду роботи з інтерактивним посібником.

2.7. Експорт та створення інсталяційного пакета для застосунку

Експорт та пакетування застосунку, створеного на основі Electron, є важливими кроками, які забезпечують готовність програми до розповсюдження серед кінцевих користувачів [16]. Нижче представлено детальний опис процесу пакування застосунку для операційних систем Windows, macOS та Linux.

Першим кроком є встановлення electron-builder, інструменту, який допомагає створювати готові до розповсюдження інсталяційні пакети з Electron-застосунків.

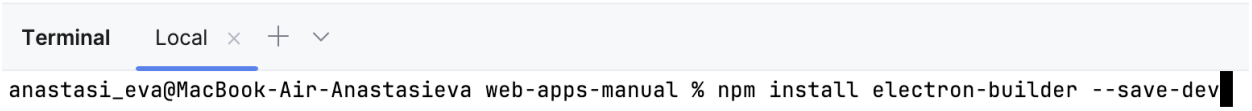
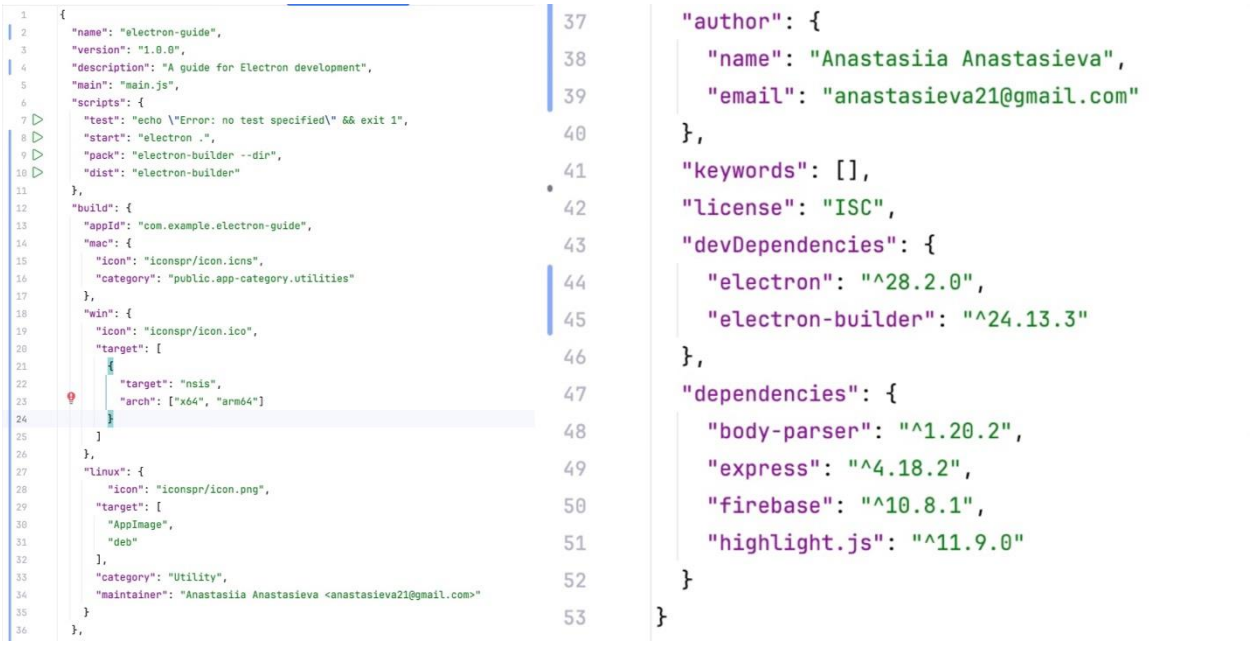


Рис. 2.28. Команда встановлення electron-builder

Для підготовки проєкту до збірки оновлюємо файл package.json, додаючи специфікації для побудови та конфігурації цілей різних платформ:



```
{
  "name": "electron-guide",
  "version": "1.0.0",
  "description": "A guide for Electron development",
  "main": "main.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "electron .",
    "pack": "electron-builder --dir",
    "dist": "electron-builder"
  },
  "build": {
    "appId": "com.example.electron-guide",
    "mac": {
      "icon": "iconspr/icon.icns",
      "category": "public.app-category.utilities"
    },
    "win": {
      "icon": "iconspr/icon.ico",
      "target": [
        {
          "target": "nsis",
          "arch": ["x64", "arm64"]
        }
      ]
    },
    "linux": {
       "icon": "iconspr/icon.png",
      "target": [
        "AppImage",
        "deb"
      ],
      "category": "Utility",
      "maintainer": "Anastasiia Anastasieva <anastasieva21@gmail.com>"
    }
  },
  "author": {
    "name": "Anastasiia Anastasieva",
    "email": "anastasieva21@gmail.com"
  },
  "keywords": [],
  "license": "ISC",
  "devDependencies": {
    "electron": "^28.2.0",
    "electron-builder": "^24.13.3"
  },
  "dependencies": {
    "body-parser": "^1.20.2",
    "express": "^4.18.2",
    "firebase": "^10.8.1",
    "highlight.js": "^11.9.0"
  }
}
```

Рис. 2.29. Код оновленого файлу package.json

Наступний крок це запуск збірки інсталяційних файлів для всіх підтримуваних платформ, а також налаштування іконок окремо для кожної платформи. Встановлення іконки відбувається в main.js

```
// Імпортування необхідних модулів з Electron та Node.js
const {app : App , BrowserWindow} = require('electron');
const path : PlatformPath | path  = require('path');

// Функція для створення основного вікна браузера
1+ usages   anastasiia-anastasieva *
function createWindow() : void  {
    // Створення вікна з певними розмірами та вебналаштуваннями
    const mainWindow : BrowserWindow  = new BrowserWindow( options: {
        width: 1200, // Ширина вікна у пікселях
        height: 714, // Висота вікна у пікселях
        icon: path.join(__dirname, 'iconspr/icon'), // Іконка вікна
        webPreferences: {
            nodeIntegration: true, // Дозволяємо використання Node.js
            contextIsolation: false // Відключаємо ізоляцію контексту для використання require
        }
    });
```

Рис. 2.30. Код встановлення іконки в main.js

У результаті виконання цих кроків було отримано інсталяційні пакети для Windows (.exe), macOS (.dmg) та Linux (.AppImage та .deb) у теці dist мого проєкту. Тепер користувачі можуть легко встановлювати Electron застосунок "Інтерактивний посібник з веброзробки" на своїх комп'ютерах, просто запустивши відповідний інсталяційний файл.

РОЗДІЛ 3. ТЕСТУВАННЯ ТА ПРАКТИЧНЕ ЗАСТОСУВАННЯ

3.1. Оцінка повноти розв’язання поставленої задачі

Початковою задачею було розробити інтерактивний посібник з програмування вебзастосунків на основі JavaScript-фреймворку Electron. Цей посібник повинен був забезпечити користувачів структурованим навчальним контентом, практичними завданнями, можливістю проходження тестів та відстеження свого прогресу. Крім того, передбачалося надати персоналізований досвід завдяки авторизації та особистому кабінету, а також забезпечити зручний офлайн-доступ.

У процесі розробки інтерактивного посібника було успішно вирішено наступні підзадачі:

1) Реалізовано систему реєстрації та авторизації користувачів з використанням Firebase Authentication.

2) Створено особистий кабінет користувача, де відображається інформація про користувача, прогрес навчання, доступ до історії пройдених тестів та аналізу помилок.

3) Впроваджено функціонал створення нотаток для полегшення процесу навчання.

4) Розроблено структурований навчальний контент, який охоплює різні аспекти веброзробки, такі як HTML, CSS, JavaScript, фреймворки, розробка на стороні сервера, API, безпека вебзастосунків та розгортання.

5) Реалізовано систему тестування, що дозволяє користувачам проходити тести для оцінки рівня засвоєння матеріалу та отримувати зворотний зв'язок щодо допущених помилок.

6) Забезпечено кросплатформну сумісність застосунку завдяки використанню Electron, що дозволяє запускати його на Windows, macOS та Linux.

7) Впроваджено можливість вибору між світлою та темною темами оформлення для покращення зручності використання.

Загалом, розроблений інтерактивний посібник є потужним інструментом для самонавчання та вдосконалення навичок веброзробки, який забезпечує зручний доступ до структурованого контенту, практичних завдань та персоналізованого досвіду. Проте, для підтримання актуальності та корисності посібника в майбутньому, необхідно продовжувати роботу над подальшим розвитком та оптимізацією його функціоналу, враховуючи зворотний зв'язок від користувачів та нові тенденції в галузі інформаційних технологій.

3.2. Перевірка вірогідності отриманих результатів

Для забезпечення того, що інтерактивний посібник повністю задовольняє визначені вимоги, було здійснено ретельну верифікацію всіх функцій та можливостей протягом процесу розробки та після його завершення. Нижче наведено перелік тестових сценаріїв, які використовувалися для перевірки виконання кожної функціональної вимоги:
Реєстрація та автентифікація користувачів:

1) Реєстрація та автентифікація користувачів:

а) Перевірка можливості створення нового облікового запису з дійсною електронною адресою та паролем.

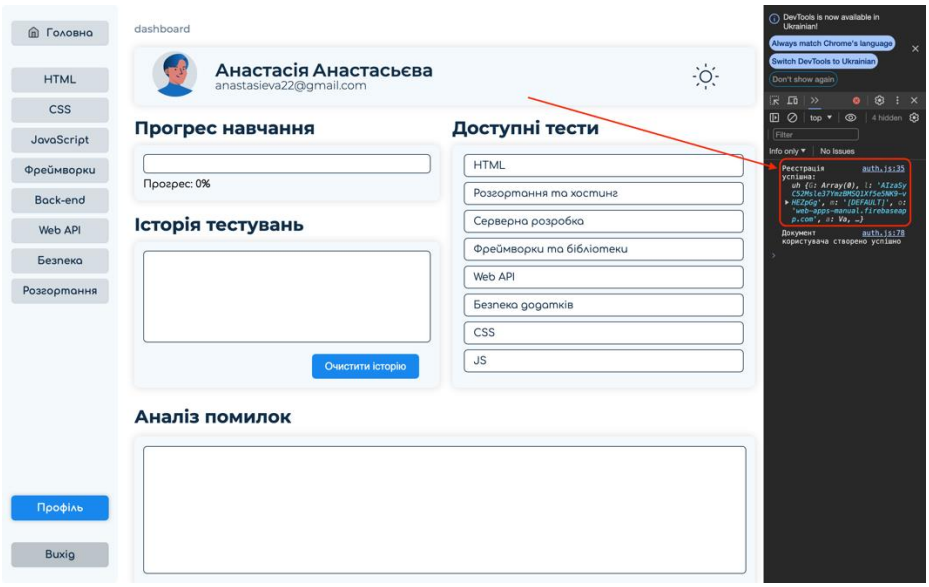


Рис. 3.1. Підтвердження успішної реєстрації

б) Під час реєстрації реалізовано належну валідацію даних та захист від загроз безпеці.

в) Перевірка можливості автентифікації наявного користувача з правильними обліковими даними.

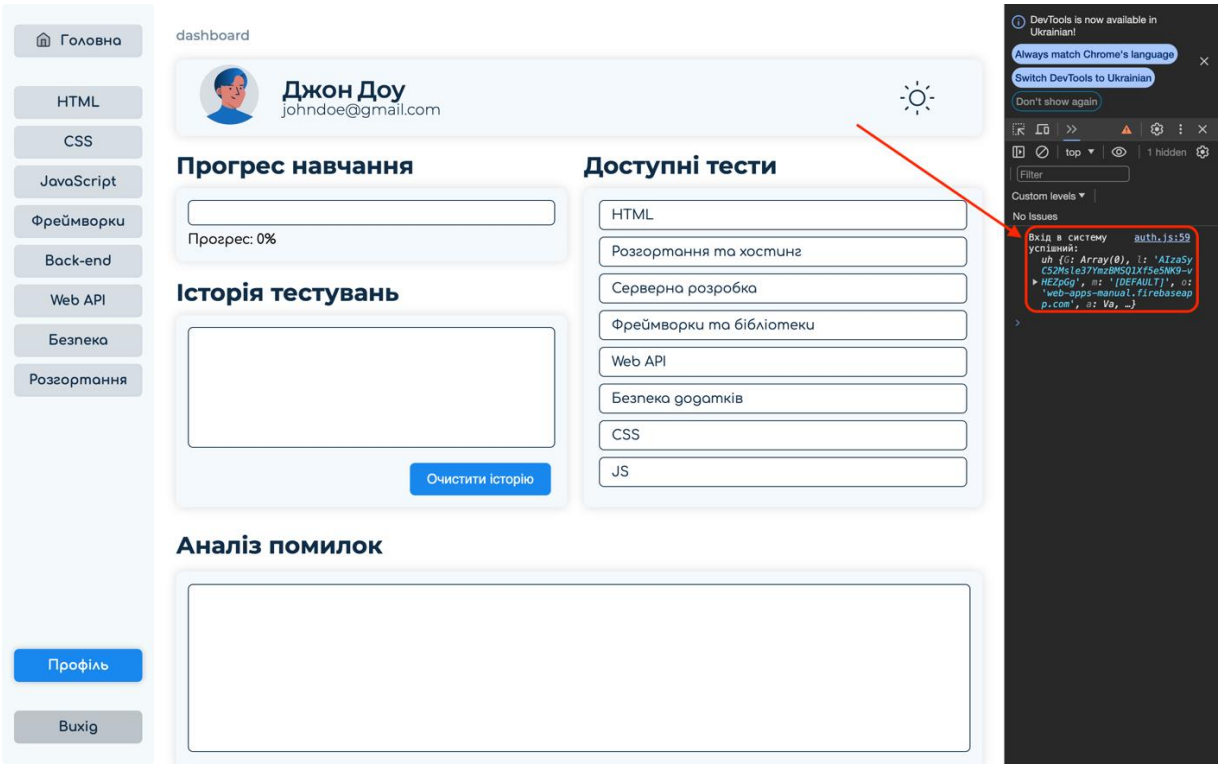


Рис. 3.2. Результат тестування успішного входу

г) При невдалій спробі автентифікації система надає відповідні повідомлення про помилку.

2) Перегляд навчального контенту:

а) Перевірка наявності та правильного відображення всіх розділів і підрозділів навчального контенту.

б) Теоретичний матеріал, приклади коду та практичні завдання відображаються належним чином.

в) Перевірка наявності механізму для простої навігації між розділами та підрозділами.

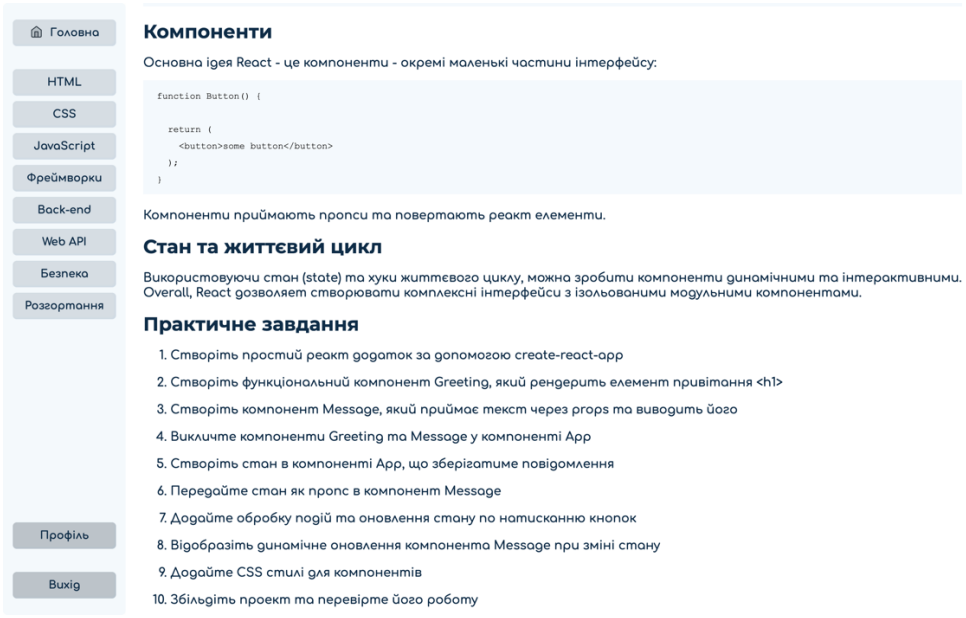


Рис. 3.3. Коректне відображення навчального контенту

3) Проходження тестів та перевірка результатів:

а) Перевірка наявності тестових завдань для кожного розділу.

б) Користувачі можуть успішно проходити тести та отримувати правильні результати.

в) Перевірка, що система надає зрозумілий зворотний зв'язок щодо допущених помилок та правильних відповідей.

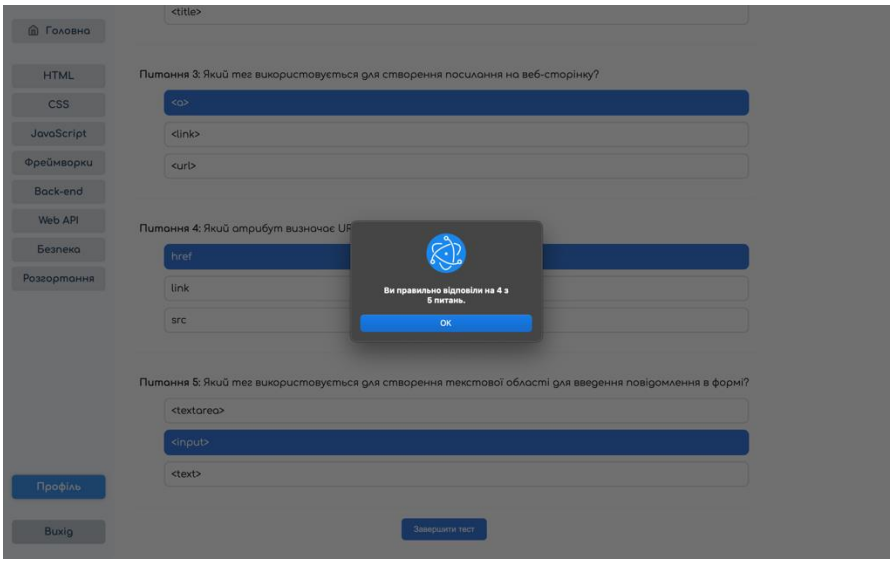


Рис. 3.4. Коректне завершення тесту

4) Створення нотаток:

а) Перевірка можливості створення, редагування та видалення нотаток користувачами.

б) Нотатки зберігаються належним чином та доступні для перегляду після перезавантаження застосунку.

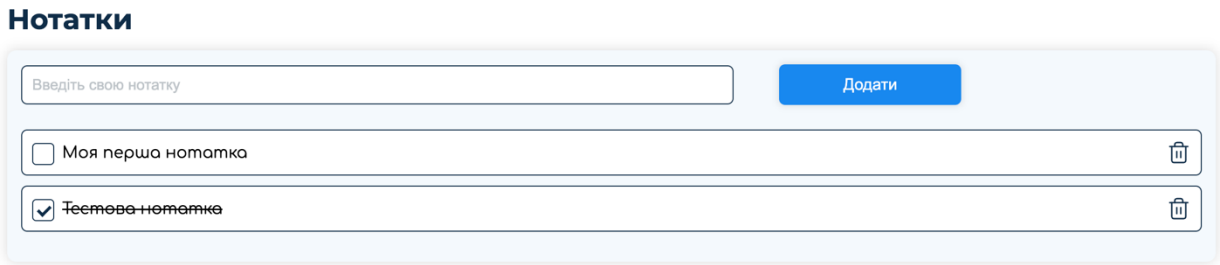


Рис. 3.5. Коректне відображення нотаток

5) Доступ до особистого кабінету:

а) Перевірка наявності та правильного відображення імені користувача в особистому кабінеті.

б) Прогрес навчання відстежується та відображається коректно.

в) Перевірка доступності історії пройдених тестів та аналізу допущених помилок.

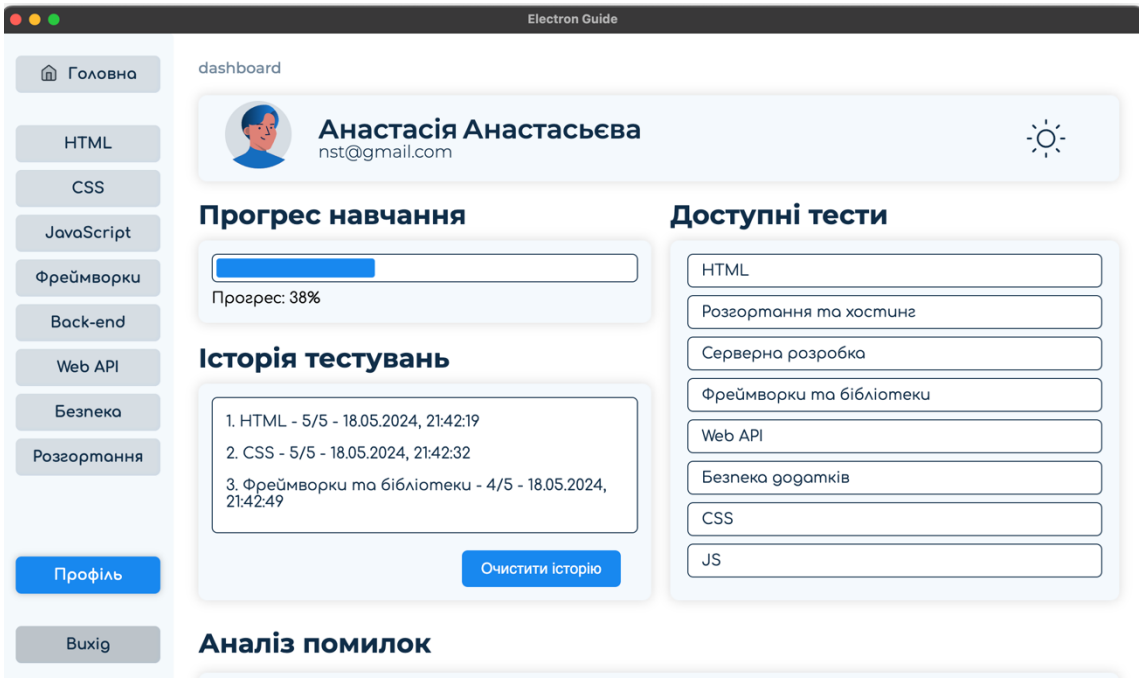


Рис. 3.6. Коректне відображення особистого кабінету (Особисті дані користувача, Прогрес навчання, Тести, Історія тестувань)

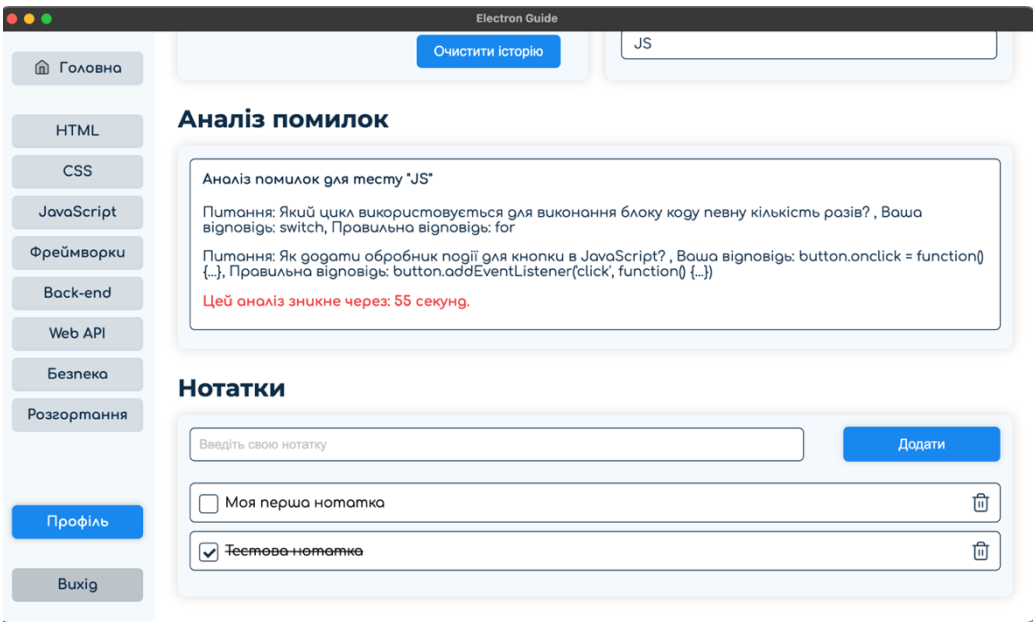


Рис. 3.7. Коректне відображення особистого кабінету (Аналіз помилок, Нотатки)

Отже, регулярне тестування дозволило своєчасно виявляти та виправляти будь-які недоліки чи помилки, забезпечуючи безперебійний досвід роботи з посібником. Ретельна перевірка реєстрації та автентифікації користувачів, перегляду навчального контенту, проходження тестів, створення нотаток та доступу до особистого кабінету гарантувала, що застосунок надає очікувані можливості та функціональність, задовольняючи потреби цільової аудиторії.

Перевірка правдивості отриманих результатів допомогла забезпечити відповідність інтерактивного посібника початковим цілям розробки. Лише після ретельної перевірки та підтвердження виконання усіх вимог можна вважати проєкт успішно завершеним і готовим до використання. Проведена верифікація є невіддільною частиною процесу розробки, яка гарантує високу якість, надійність та відповідність інтерактивного посібника поставленим завданням.

3.3. Порівняння результатів з наявними аналогами

Порівняно з аналогами, інтерактивний посібник на основі Electron має низку суттєвих переваг. По-перше, порівняно з вебзастосунками та онлайн-платформами, такими як Learn Git Branching та Codecademy, інтерактивний

посібник на Electron забезпечує можливість працювати без підключення до Інтернету, що є вкрай важливим для багатьох користувачів. Крім того, цей застосунок пропонує повноцінне персоналізоване середовище з особистим кабінетом, де користувачі можуть відстежувати свій прогрес, аналізувати допущені помилки, переглядати історію проходження тестів та вести власні нотатки. Це значно покращує досвід навчання та допомагає користувачам ефективно керувати своїм процесом розвитку.

На відміну від Programming Hub та Java Recipes, інтерактивний посібник поєднує теоретичні матеріали, приклади коду та практичні вправи в одному місці, забезпечуючи комплексний підхід до вивчення веброзробки. Крім того, він охоплює широкий спектр тем, включаючи HTML, CSS, JavaScript, фреймворки, технології на стороні сервера та інші важливі аспекти вебпрограмування. Це дозволяє користувачам отримати ґрунтовні знання та практичні навички, необхідних для успішної кар'єри розробника застосунків [18].

Ще однією важливою перевагою є підтримка локалізації, зокрема української мови. Це робить застосунок доступним для українських користувачів та полегшує процес навчання для тих, хто віддає перевагу рідній мові.

Великою перевагою над іншими аналогами є те що, застосунок пропонує можливість вести власні нотатки та персоналізувати досвід в особистому кабінеті. Це дозволяє користувачам адаптувати навчальний процес відповідно до своїх потреб та уподобань.

Підсумовуючи, інтерактивний посібник на основі JavaScript-фреймворку Electron поєднує переваги настільного застосунку, навчального контенту, персоналізації, прогресу та української локалізації, відрізняючись від Learn Git Branching, Codecademy, Level Up Tutorials, SoloLearn, Code: Learn Coding/Programming, Programming Hub комплексністю, структурованістю та підтримкою рідної мови для українських користувачів.

ВИСНОВКИ

У процесі виконання даної кваліфікаційної роботи була успішно досягнута головна мета - розробка комплексного інтерактивного посібника на основі JavaScript-фреймворку Electron для навчання розробки вебзастосунків. Створений застосунок поєднує теоретичні знання, практичні завдання та інтерактивні елементи в єдиному зручному інтерфейсі, забезпечуючи кросплатформну сумісність та багатофункціональність.

Для досягнення поставленої мети був проведений аналіз предметної області, що дозволив виявити потенціал у створенні інтерактивного посібника з персоналізованим досвідом, структурованим контентом та практичними вправами. Вдалий вибір технологій та інструментів, таких як Electron, HTML, CSS, JavaScript та допоміжні інструменти (Firebase, Figma), забезпечив створення високофункціонального та зручного у використанні застосунку.

В ході розробки була створена архітектура та структура застосунку, що включає модулі авторизації, особистого кабінету, тестування та управління навчальним контентом. Безпечна автентифікація користувачів та зберігання їх даних забезпечується завдяки інтеграції з Firebase. Проведено ретельне тестування функціональних вимог та порівняння результатів з наявними аналогами, що підтвердило переваги розробленого рішення.

Підсумовуючи, у ході виконання роботи були успішно вирішені всі поставлені завдання, включаючи розробку кросплатформного настільного застосунку на основі Electron, інтеграцію системи авторизації та збереження даних користувачів за допомогою Firebase, створення інтуїтивно зрозумілого інтерфейсу користувача та забезпечення інтерактивності навчального контенту. Таким чином, досягнута головна мета роботи - створення інтерактивного посібника, який забезпечує ефективне, зручне та персоналізоване навчання програмування для україномовних користувачів, сприяючи підвищенню якості освіти в ІТ-галузі, популяризації програмування серед широкої аудиторії та розширенню можливостей для професійного розвитку фахівців у сфері веброзробки.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Пасічнік О.Г., Пасічнік О.В., Стеценко І.В. Основи веб-дизайну: Вид. група BHV, 2009. 336 с.
2. Agarwal R. Firebase for Flutter Developers: Authentication, Database and Storage Mastery: 2023. 130 p.
3. Dennis. D. Beginning Progressive Web App Development: Apress, 2017. 286p.
4. Griffith C. Electron: From Beginner to Pro: Learn to Build Cross Platform Desktop Applications using Github's Electron: Apress, 2017. 282 p.
5. Hostmann C. S. Modern JavaScript for the Impatient: Addison-Wesley Professional, 2020. 352 p.
6. Jensen P. Cross-Platform Desktop Applications: Using Node, Electron, and NW.js First Edition: New York, 2017. 312p.
7. Kinney S. Electron in Action First Edition: Manning, 2018. 376 p.
8. Melehi D. Developing Desktop Applications with Electron: Independently Published, 2023. 64 p.
9. Rappl F. Modern Frontend Development with Node.js: A compendium for modern JavaScript web development within the Node.js ecosystem: Packt Publishing, 2022. 208 p.
10. Saunder. A. Building Cross-Platform Apps using Titanium, Alloy, and Appcelerator Cloud Services: O'Reilly Media, 2014. 361p.
11. Tanna M. Serverless Web Applications with React and Firebase: Develop real-time applications for web and mobile platforms: Packt Publishing, 2018. 286 p.
12. Wieruch R. The Road to Firebase: Your journey to master Firebase in JavaScript: 2021. 201 p.
13. Yahiaoui H. Firebase Cookbook: Over 70 recipes to help you create real-time web and mobile applications with Firebase: Packt Publishing, 2017. 290p.
14. Electron | GitHub. URL: https://github.com/electron (дата звернення 20.02.2024)

15. Electron | Build cross-platform desktop apps with JavaScript, HTML, CSS. URL: https://www.electronjs.org/ (дата звернення:25.02.2024).

16. Electron Fiddle. URL: https://www.electronjs.org/fiddle (дата звернення:26.02.2024).

17. Introduction to Node.js. URL: https://nodejs.org/en/learn/getting-started/introduction-to-nodejs (дата звернення:20.03.2024).

18. Learn web development | MDW. URL: https://developer.mozilla.org/en-US/docs/Learn (дата звернення:20.03.2024).

19. Questions tagged [electron] - StackOverflow. URL: https://stackoverflow.com/questions/tagged/electron(дата звернення:22.03.2024).

20. The best JS frameworks for desktop applications - Luxnet. URL: https://luxnet.io/uk/blog/the-best-of-js-frameworks-for-desktop-applications (дата звернення:23.04.2024).

ДОДАТКИ

Графічні матеріали

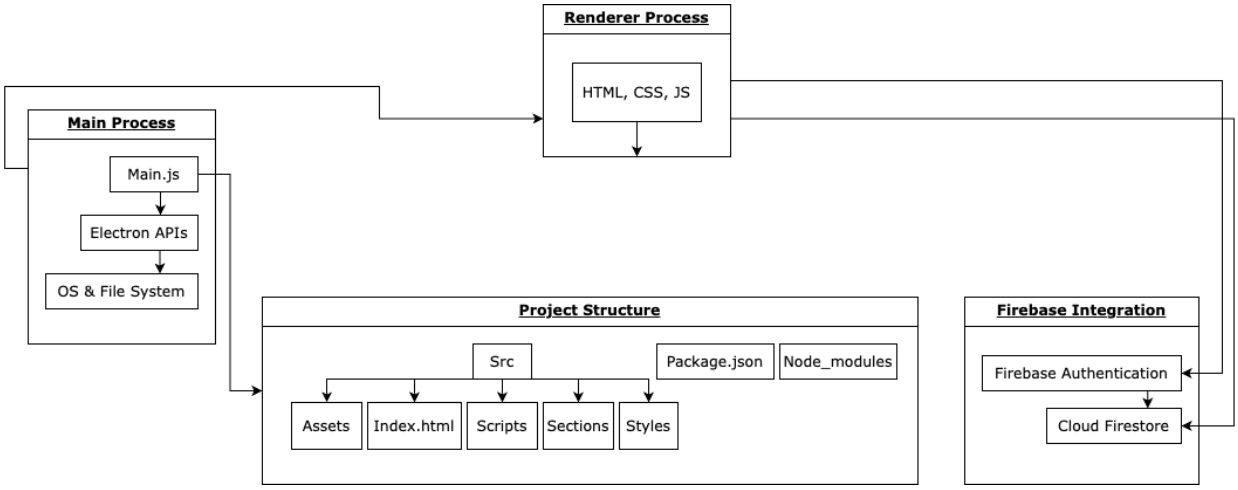


Рис. А.1. Діаграма архітектури застосунку

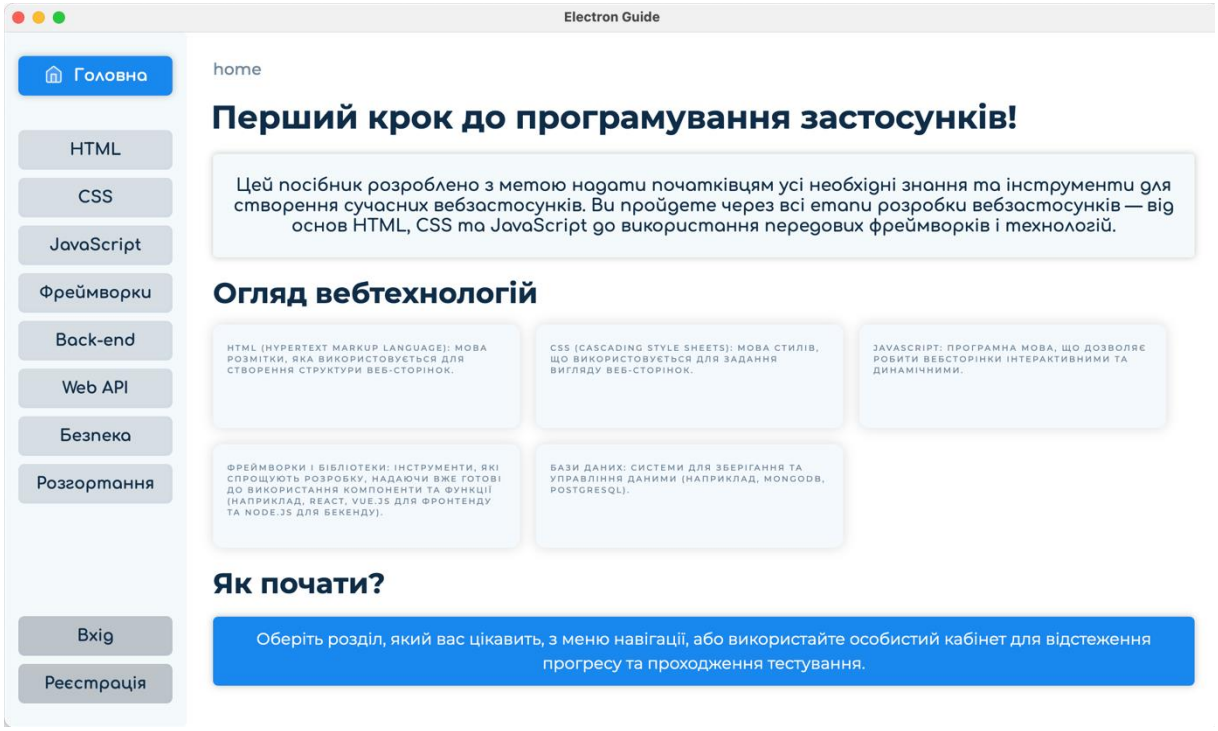


Рис. А.2. Головний екран застосунку

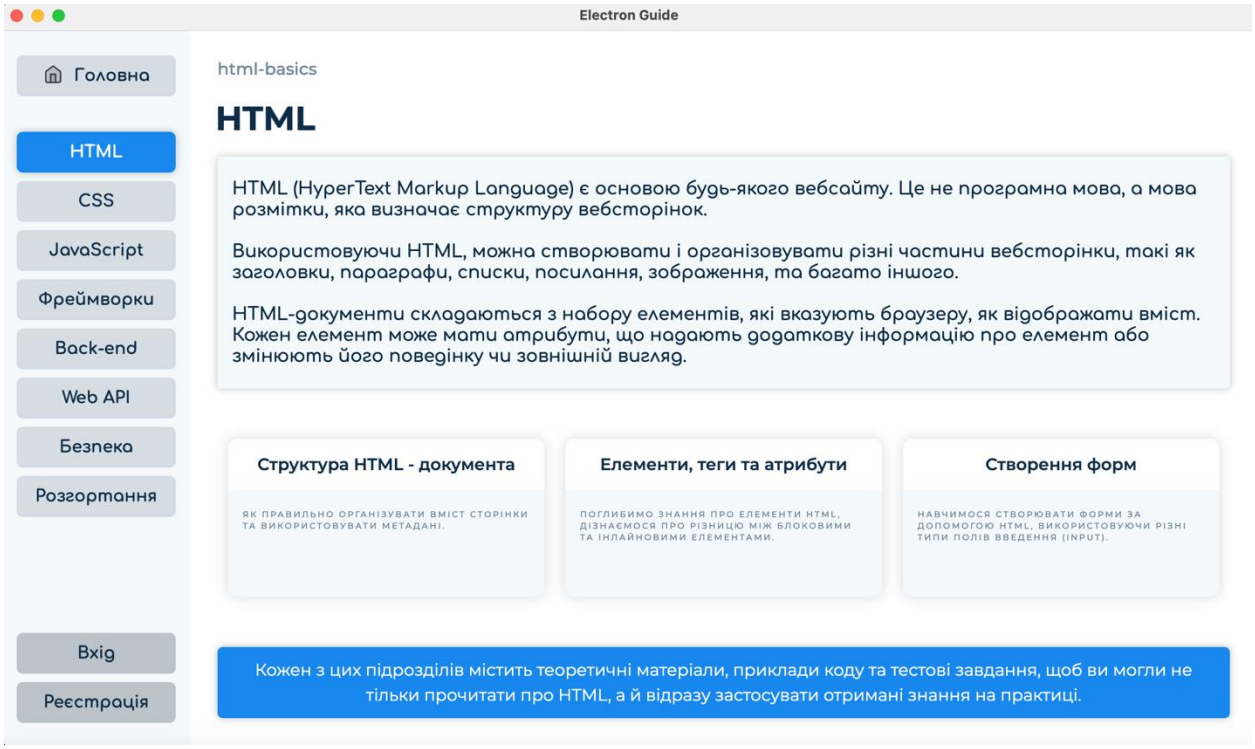


Рис. А.3. Розділ в застосунку

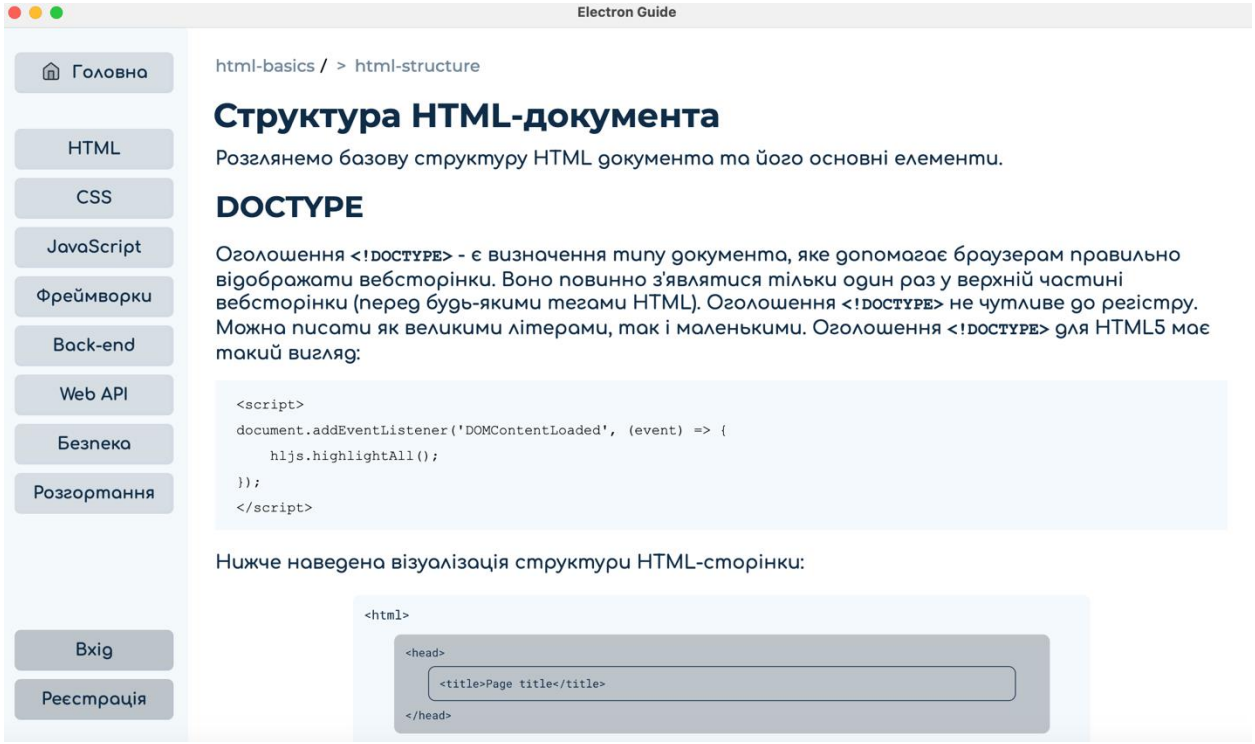


Рис. А.4. Підрозділ в застосунку

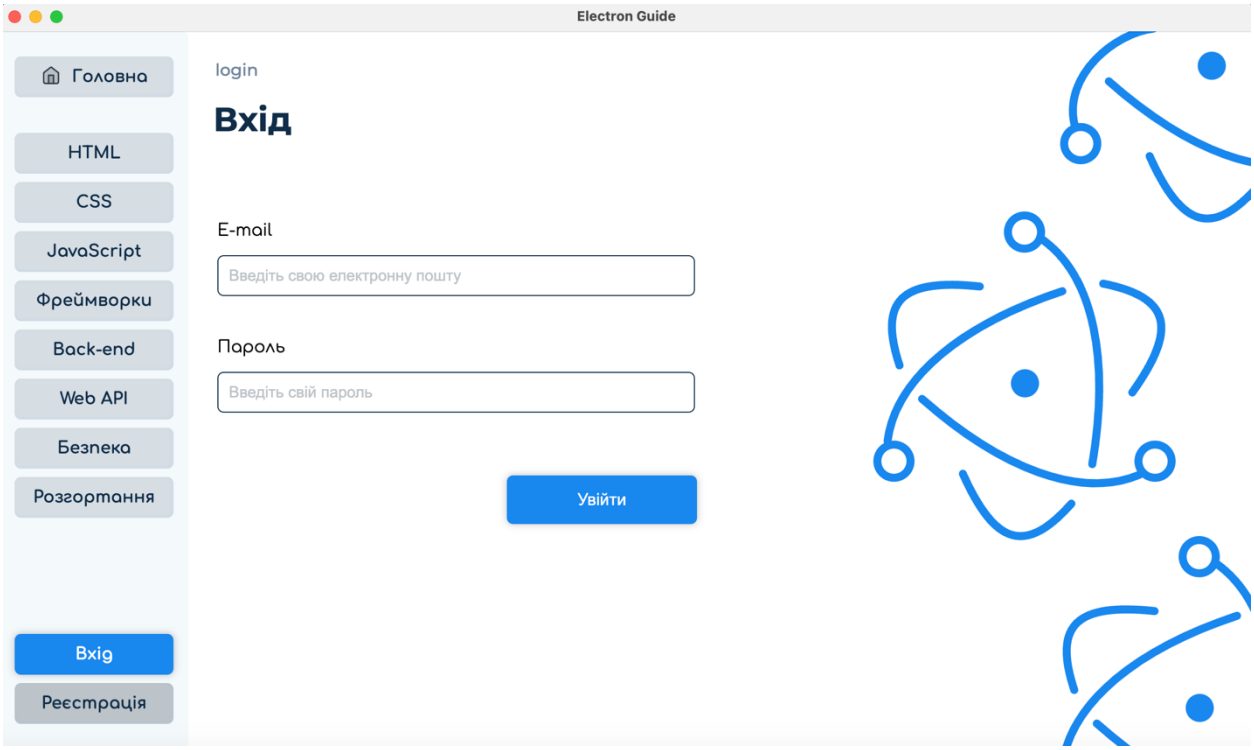


Рис. А.5. Екран авторизації

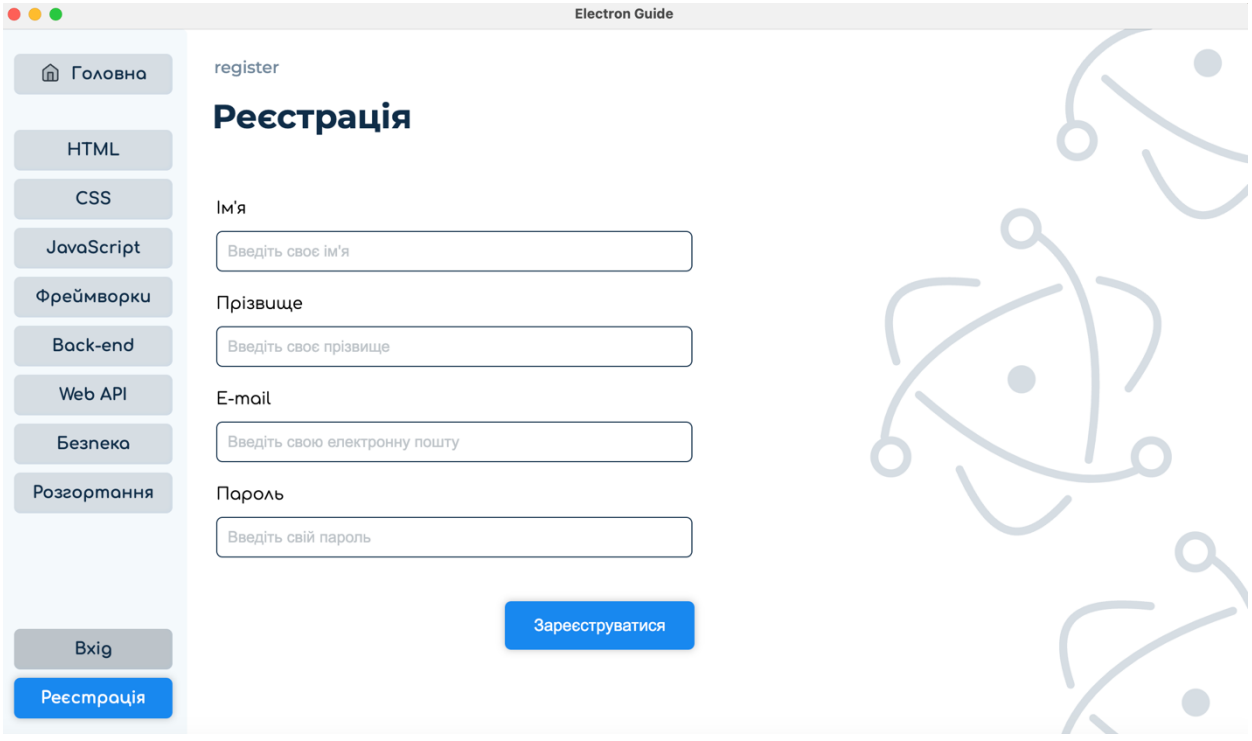


Рис. А.6. Екран реєстрації

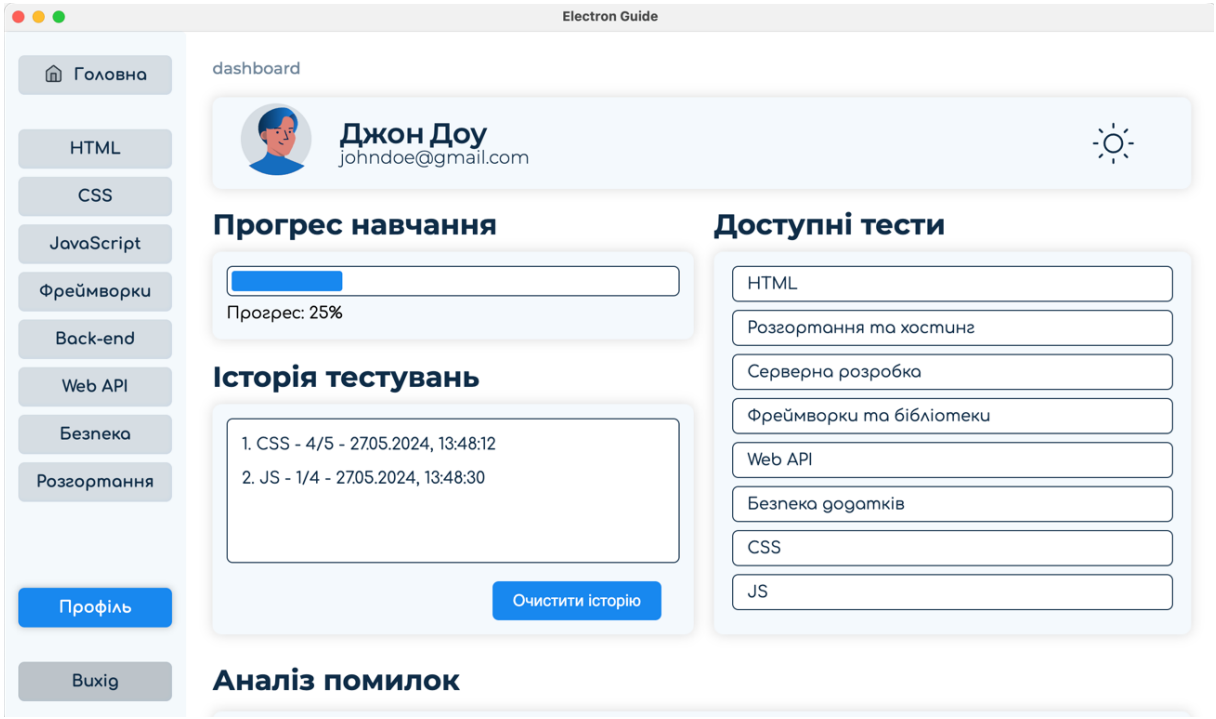


Рис. А.7. Екран профілю користувача, частина перша

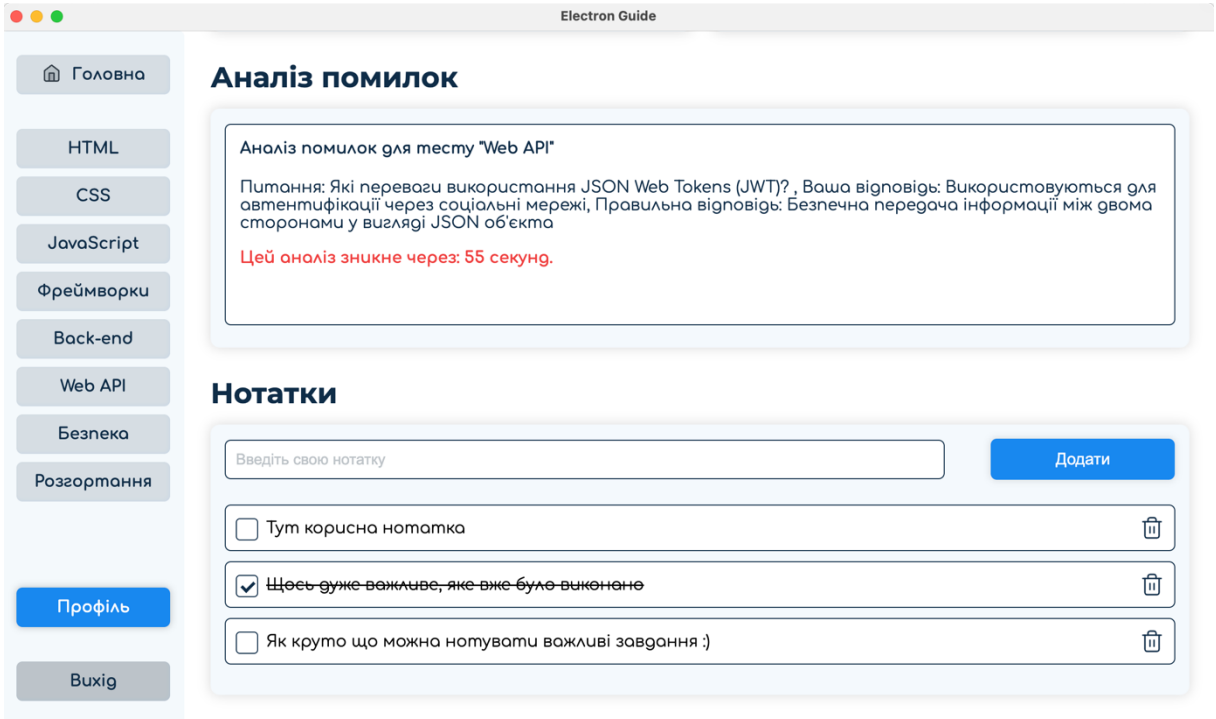


Рис. А.8. Екран профілю користувача, частина друга

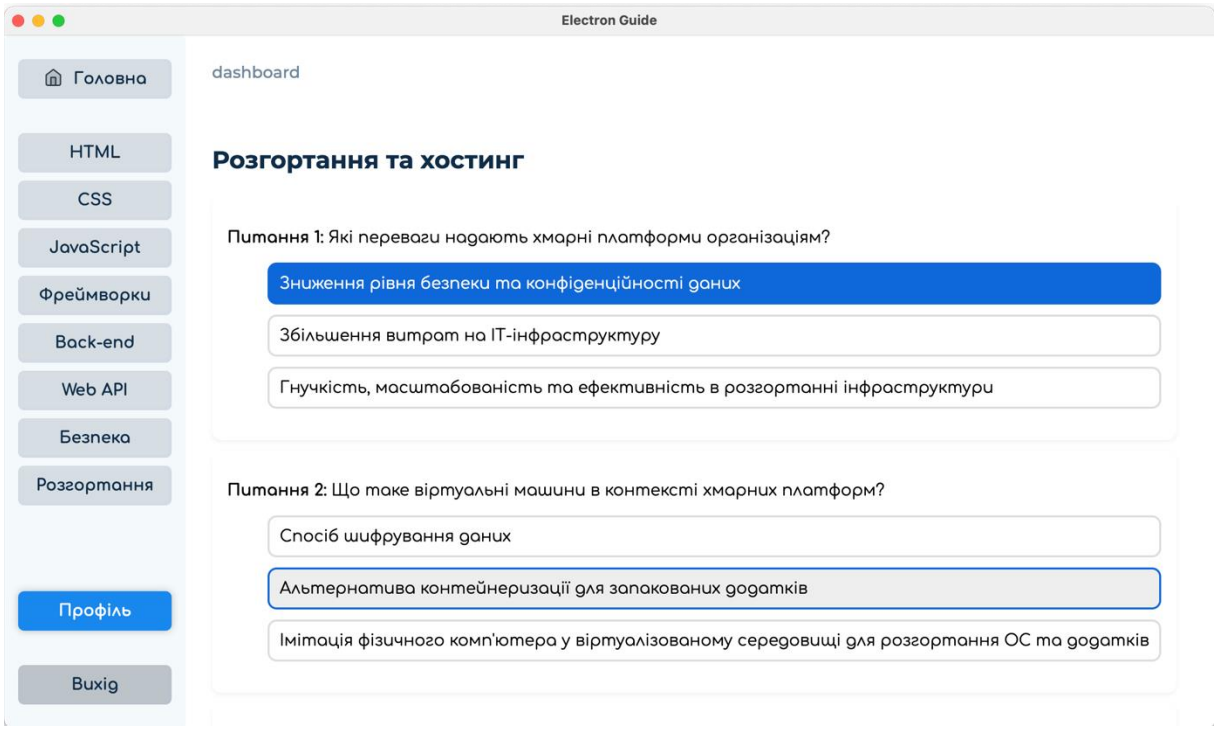


Рис. А.9. Приклад екрану тестування

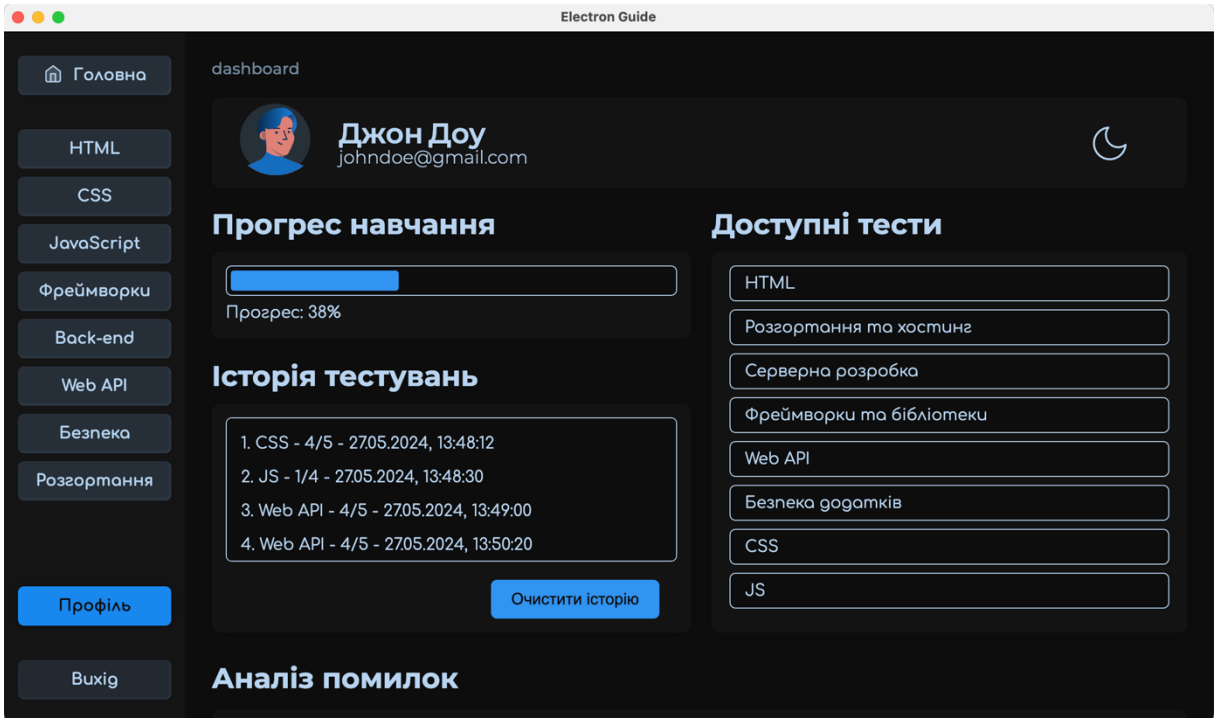


Рис. А.10. Приклад темної теми застосунка

Лістинг застосунку

main.js:

```
// Імпортування необхідних модулів з Electron та Node.js
const {app, BrowserWindow} = require('electron');
const path = require('path');
// Функція для створення основного вікна браузера
function createWindow() {
    // Створення вікна з певними розмірами та вебналаштуваннями
    const mainWindow = new BrowserWindow({
        width: 1200, // Ширина вікна у пікселях
        height: 714, // Висота вікна у пікселях
        icon: path.join(__dirname, 'iconspr/icon'), // Іконка вікна
        webPreferences: {
            nodeIntegration: true, // Дозволяємо використання Node.js
            contextIsolation: false // Відключаємо ізоляцію контексту для
використання require
        }
    });
    // Завантаження index.html у вікно.
    mainWindow.loadFile(path.join(__dirname, 'src/index.html'));
    // Побудова шляху до CSS файлу
    const cssPath = path.join(__dirname, 'styles', 'main.css');
    console.log(cssPath); // Виведіть шлях до консолі для перевірки
    // Відкриття DevTools.
    // mainWindow.webContents.openDevTools();
}
// Цей метод буде викликаний, коли Electron завершить ініціалізацію та
готовий створювати вікна браузера.
// Деякі API можуть використовуватися лише після цього події.
app.whenReady().then(createWindow);
// Виходити з програми, коли всі вікна закриті, крім MacOS, де це звичайна
поведінка.
app.on('window-all-closed', () => {
    if (process.platform !== 'darwin') {
        app.quit();
    }
});
// Обробник події активації застосунку, наприклад, після виходу зі стану
"dock" на MacOS
app.on('activate', () => {
    // На MacOS звичайно створювати вікно в додатку, коли клікають на іконку
доку і жодних інших вікон відкрито не є.
    if (BrowserWindow.getAllWindows().length === 0) {
        createWindow();
    }
});
```

src/scripts/auth.js:

```
// Слухач подій, який виконується, коли весь HTML документ повністю
завантажено і готовий
document.addEventListener('DOMContentLoaded', function () {
    // Створюємо новий спостерігач за змінами у DOM
    const observer = new MutationObserver((mutations) => {
        // Ітерація через кожну мутацію у DOM
        mutations.forEach((mutation) => {
            // Якщо у мутації є додані вузли, ініціалізуємо форми авторизації
            if (mutation.addedNodes.length) {
                initializeAuthForms();
            }
```

```
        });
    });
    // Встановлюємо спостерігача на основний контент сайту для відстеження
додавання нових елементів
    observer.observe(document.querySelector('#main-content'), {childList:
true, subtree: true});
});
// Функція для ініціалізації форм авторизації
function initializeAuthForms() {
    // Знаходимо форму реєстрації за її ID
    const registrationForm = document.querySelector('#registerForm');
    if (registrationForm) {
        // Додаємо обробник подій на подію 'submit'
        registrationForm.addEventListener('submit', function (e) {
            e.preventDefault(); // Зупиняємо стандартну обробку події
            // Отримуємо значення з полів форми
            const email = document.getElementById('email').value;
            const password = document.getElementById('password').value;
            const firstName = document.getElementById('first-name').value; //
Отримання значення імені
            const lastName = document.getElementById('lastname').value; //
Отримання значення прізвища
            // Викликаємо Firebase API для створення нового користувача
            firebase.auth().createUserWithEmailAndPassword(email, password)
                .then((userCredential) => {
                    console.log("Реєстрація успішна:", userCredential.user);
                    // Додаємо збереження імені та прізвища
                    createUserDocument(userCredential.user.uid, email,
firstName, lastName);
                    window.loadSection('dashboard');
                })
                .catch((error) => {
                    console.error("Помилка реєстрації:", error.code,
error.message);
                    document.getElementById('registerError').textContent =
error.message;
                });
        });
    }
    // Знаходимо форму входу за її ID
    const loginForm = document.querySelector('#loginForm');
    if (loginForm) {
        // Обробка події входу
        loginForm.addEventListener('submit', function (e) {
            e.preventDefault();
            const email = document.getElementById('loginEmail').value;
            const password = document.getElementById('loginPassword').value;
            // Викликаємо Firebase API для входу користувача
            firebase.auth().signInWithEmailAndPassword(email, password)
                .then((userCredential) => {
                    console.log("Вхід в систему успішний:",
userCredential.user);
                    window.loadSection('dashboard');
                })
                .catch((error) => {
                    console.error("Помилка входу:", error.code,
error.message);
                    document.getElementById('loginError').textContent =
error.message;
                });
        });
    }
}
// Функція для створення запису про користувача в базі даних
```

```
function createUserDocument(userId, email, firstName, lastName) {
    // Зберігаємо інформацію про нового користувача у колекції 'users' у
Firebase Firestore
    db.collection('users').doc(userId).set({
        email: email,
        firstName: firstName,
        lastName: lastName
    }).then(() => {
        console.log("Документ користувача створено успішно");
    }).catch((error) => {
        console.error("Не вдалося створити документ користувача", error);
    });
}
```

src/scripts/dashboard.js:

```
// Відображення задач користувача
function displayTasks() {
    const userId = firebase.auth().currentUser.uid; // Отримання
ідентифікатора поточного користувача
    getTasksForUser(userId, (error, tasks) => {
        if (error) {
            console.error("Не вдалося отримати задачі:", error); // Логування
помилки, якщо задачі не можна отримати
        } else {
            const tasksListElement = document.getElementById('tasks'); //
Знаходження елементу для відображення списку задач
            tasksListElement.innerHTML = ''; // Очищуємо список перед
додаванням нових завдань
            tasks.forEach((task, index) => { // Перебір отриманих задач
                const taskElement = document.createElement('li'); //
Створення елементу списку для кожної задачі
                const taskLabel = document.createElement('label');
                taskLabel.classList.add('custom-checkbox'); // Додавання
стилю для візуального відображення чекбоксу
                const taskCheckbox = document.createElement('input');
                taskCheckbox.type = 'checkbox';
                taskCheckbox.checked = task.completed; // Встановлення
статусу задачі
                taskCheckbox.id = 'task-' + index; // Присвоєння унікального
ID для кожного чекбоксу
                const checkmark = document.createElement('span');
                checkmark.classList.add('checkmark');
                taskLabel.appendChild(taskCheckbox);
                taskLabel.appendChild(checkmark);
                // Додавання обробника події для зміни стану задачі
                taskCheckbox.addEventListener('change', () => {
                    toggleTaskCompleted(userId, index, taskCheckbox.checked);
                });
                const taskText = document.createElement('span');
                taskText.textContent = task.text; // Додавання тексту задачі
                if (task.completed) {
                    taskText.style.textDecoration = 'line-through'; //
Перекреслення тексту, якщо задача виконана
                }
                taskElement.appendChild(taskLabel);
                taskElement.appendChild(taskText);
                // Створення та додавання кнопки для видалення задачі
                const deleteButton = document.createElement('button');
                deleteButton.classList.add('delete-button');
                deleteButton.textContent = 'Видалити';
                deleteButton.addEventListener('click', () =>
deleteTask(userId, index));
                taskElement.appendChild(deleteButton);
                tasksListElement.appendChild(taskElement);
```

```
            });
        }
    });
}
// Функція для видалення задачі
function deleteTask(userId, taskIndex) {
    const userRef = db.collection('users').doc(userId);
    userRef.get().then(doc => {
        if (doc.exists) {
            let tasks = doc.data().notes;
            // Видалення задачі з масиву за індексом
            tasks.splice(taskIndex, 1);
            // Оновлення завдань в документі користувача
            userRef.update({ notes: tasks }).then(() => {
                displayTasks(); // Оновлюємо відображення завдань після
видалення
            }).catch(error => {
                console.error("Помилка при видаленні завдання:", error);
            });
        }
    }).catch(error => {
        console.error("Помилка при отриманні документу:", error);
    });
}
function setupAddTaskButton() {
    const addTaskButton = document.getElementById('add-task-button');
    if (addTaskButton) {
        addTaskButton.addEventListener('click', () => {
            const taskText = document.getElementById('new-task').value;
            if (taskText.trim() === "") return; // Ігноруємо порожні задачі
            const userId = firebase.auth().currentUser.uid;
            console.log("Adding task for user:", userId); // Додатковий лог
            addTaskForUser(userId, taskText, (error) => {
                if (error) {
                    console.error("Error adding task", error);
                } else {
                    console.log("Task added successfully");
                    document.getElementById('new-task').value = ''; //
Очищуємо поле після додавання
                    displayTasks(); // Оновлюємо список задач
                }
            });
        });
    } else {
        console.error("The add-task-button was not found.");
    }
}
function toggleTaskCompleted(userId, taskIndex, completed) {
    // Отримуємо посилання на документ користувача та завдання
    const userRef = db.collection('users').doc(userId);
    userRef.get().then((doc) => {
        if (doc.exists) {
            let tasks = doc.data().notes;
            // Оновлюємо статус завдання
            tasks[taskIndex].completed = completed;
            // Оновлюємо завдання в документі користувача
            userRef.update({ notes: tasks }).then(() => {
                // Оновлюємо відображення завдань
                displayTasks();
            }).catch((error) => {
                console.error("Помилка при оновленні завдання:", error);
            });
        }
    }).catch((error) => {
```

```
        console.error("Помилка при отриманні документу:", error);
    });
}
//Завантаження тестів з бази даних
function loadAvailableTests() {
    db.collection("tests").get().then((querySnapshot) => {
        const testsListElement = document.getElementById('tests-list');
        testsListElement.innerHTML = ''; // Очистка списку перед
завантаженням нових даних

        querySnapshot.forEach((doc) => {
            const li = document.createElement('li');
            li.textContent = doc.data().title;
            li.style.cursor = 'pointer';
            li.onclick = () => loadTest(doc.id); // Функція loadTest ще не
реалізована
            testsListElement.appendChild(li);
        });
    });
}
function loadTest(testId) {
    db.collection("tests").doc(testId).get().then((doc) => {
        if (!doc.exists) {
            console.log("Тест не знайдено!");
            return;
        }
        const test = doc.data();
        displayTest(test);
    }).catch((error) => {
        console.error("Помилка при завантаженні тесту:", error);
    });
}
function displayTest(test) {
    const mainContent = document.getElementById('main-content');
    mainContent.innerHTML = `<h3 class="test-title">${test.title}</h3>`;
    test.questions.forEach((question, index) => {
        const questionElement = document.createElement('div');
        questionElement.className = 'question';
        questionElement.innerHTML = `<p><b>Питання ${index + 1}:</b>
${question.question}</p>`;
        const optionsList = document.createElement('ul');
        question.options.forEach((option, optionIndex) => {
            const optionElement = document.createElement('li');
            optionElement.textContent = option;
            optionElement.className = 'answer-option';
            optionElement.addEventListener('click', function() {
                // Видаляємо клас 'selected' у всіх інших варіантів
відповідей цього питання
                this.parentNode.querySelectorAll('.answer-option').forEach(el
=> {
                    el.classList.remove('selected');
                });
                // Додаємо клас 'selected' до обраного варіанта відповіді
                this.classList.add('selected');
                // Зберігання вибраної відповіді (можна розширити для
подальшої обробки)
            });
            optionsList.appendChild(optionElement);
        });
        questionElement.appendChild(optionsList);
        mainContent.appendChild(questionElement);
    });
    // Кнопка для завершення тесту
    const submitButton = document.createElement('button');
```

```
    submitButton.textContent = 'Завершити тест';
    submitButton.className = 'submit-button';
    // Доданий обробник подій
    // Виклик функції з параметром test
    submitButton.addEventListener('click', function() {
        let score = 0;
        const details = [];
        test.questions.forEach((question, index) => {
            const selectedOption = mainContent.querySelectorAll('.answer-
option.selected')[index];
            const isCorrect = selectedOption && selectedOption.textContent
=== question.correctAnswer;
            if (isCorrect) {
                score++;
            }
            details.push({
                question: question.question,
                userAnswer: selectedOption ? selectedOption.textContent :
null,
                correctAnswer: question.correctAnswer,
                isCorrect: isCorrect
            });
        });
        // Перевірка, чи всі питання були відповідені
        if(details.some(detail => detail.userAnswer === null)) {
            alert('Будь ласка, дайте відповідь на всі питання.');
            return; // Не завершуємо тест, якщо є питання на які не дали
відповідь
        }
        alert(`Ви правильно відповіли на ${score} з ${test.questions.length}
питань.`);
        saveTestResult(score, test.questions.length, test.title, details);
        loadSection('dashboard'); // Перехід до особистого кабінету після
закриття alert
    });
    mainContent.appendChild(submitButton);
}
// Приклад функції для збереження результату тесту
function saveTestResult(score, totalQuestions, testTitle, details) {
    const userId = firebase.auth().currentUser.uid;
    // Визначення часу, коли аналіз помилок стане недоступним (наприклад,
через 5 хвилин)
    const analysisEndTime = new Date().getTime() + 300000;
    const result = {
        testTitle: testTitle,
        score: score,
        totalQuestions: totalQuestions,
        timestamp: new Date(),
        details: details, // Збереження деталей відповідей
        analysisEndTime: analysisEndTime // Час закінчення аналізу помилок
    };
    const userRef = db.collection('users').doc(userId);
    userRef.update({
        testsCompleted: firebase.firestore.FieldValue.arrayUnion(result)
    })
        .then(() => {
            console.log("Результат тесту з деталями та часом аналізу успішно
збережено.");
            updateLearningProgress(); // Оновлення прогресу навчання на
сторінці
            displayErrorAnalysis();
        })
        .catch((error) => {
            console.error("Помилка збереження результату тесту з деталями:",
```

```
error);
        });
}
// Відображення історії тестів
function displayTestHistory() {
    const userId = firebase.auth().currentUser.uid;
    const userRef = db.collection('users').doc(userId);
    userRef.get().then((doc) => {
        if (doc.exists) {
            const userData = doc.data();
            const historyListElement = document.getElementById('history-
list');
            historyListElement.innerHTML = ''; // Очистити поточний список
            userData.testsCompleted.forEach((testResult, index) => {
                const li = document.createElement('li');
                li.textContent = `${index + 1}. ${testResult.testTitle} -
${testResult.score}/${testResult.totalQuestions} -
${testResult.timestamp.toDate().toLocaleString()}`;
                historyListElement.appendChild(li);
            });
            updateLearningProgress(); // Оновлення прогресу навчання на
сторінці
            attachClearHistoryHandler();
        }
    }).catch((error) => {
        console.error("Помилка отримання даних користувача:", error);
    });
}
function attachClearHistoryHandler() {
    const clearHistoryButton = document.getElementById('clear-history-
button');
    if (clearHistoryButton) {
        clearHistoryButton.addEventListener('click', clearTestHistory);
    } else {
        console.error('Button not found');
    }
}
function clearTestHistory() {
    const userId = firebase.auth().currentUser.uid;
    const userRef = db.collection('users').doc(userId);
    userRef.update({
        testsCompleted: [] // Очистка історії
    })
        .then(() => {
            console.log("Історія тестувань успішно очищена.");
            displayTestHistory(); // Поновлення історії тестувань на сторінці
        })
        .catch((error) => {
            console.error("Помилка при очищенні історії тестувань:", error);
        });
}
function displayErrorAnalysis() {
    const userId = firebase.auth().currentUser.uid;
    db.collection('users').doc(userId).get().then((doc) => {
        if (doc.exists) {
            const userData = doc.data();
            // Отримання останнього результату тесту
            const latestTestResult =
userData.testsCompleted[userData.testsCompleted.length - 1];
            if (latestTestResult && latestTestResult.details) {
                // Перевірка, чи не минув час закінчення аналізу
                if (latestTestResult.analysisEndTime <= new Date().getTime())
{
                    // Якщо час закінчення аналізу минув, не відображаємо
```

```
аналіз
                    document.getElementById('analysis-content').innerHTML =
'';
                    document.getElementById('error-analysis-timer').innerHTML
= '';
                    return;
                }
                // Фільтруємо неправильні відповіді
                const incorrectAnswers =
latestTestResult.details.filter(detail => !detail.isCorrect);
                if (incorrectAnswers.length > 0) {
                    // Якщо є неправильні відповіді, виводимо аналіз і таймер
                    const analysisContentElement =
document.getElementById('analysis-content');
                    analysisContentElement.innerHTML = `<h4>Аналіз помилок
для тесту "${latestTestResult.testTitle}"</h4>`;
                    incorrectAnswers.forEach(answer => {
                        const answerElement = document.createElement('p');
                        answerElement.innerHTML = `Питання:
${answer.question}, Ваша відповідь: ${answer.userAnswer || "не вказано"},
Правильна відповідь: ${answer.correctAnswer}`;
                        analysisContentElement.appendChild(answerElement);
                    });
                    // Встановлення таймера
                    setupErrorAnalysisTimer(60,
document.getElementById('error-analysis-timer'), analysisContentElement);
                } else {
                    // Якщо всі відповіді правильні, очищуємо контент аналізу
помилок і не виводимо таймер
                    document.getElementById('analysis-content').innerHTML =
'';
                    document.getElementById('error-analysis-timer').innerHTML
= '';
                }
            }
        }
    }).catch(error => {
        console.error("Помилка отримання аналізу помилок:", error);
    });
}
function setupErrorAnalysisTimer(duration, timerElement,
analysisContentElement) {
    let startTime = localStorage.getItem('timerStart');
    if (!startTime) {
        startTime = new Date().getTime();
        localStorage.setItem('timerStart', startTime);
    }
    let timeLeft = Math.max(0, Math.floor(duration - (new Date().getTime() -
startTime) / 1000));
    if (analysisContentElement.innerHTML === '') {
        timerElement.innerHTML = '';
        return;
    }
    timerElement.textContent = `Цей аналіз зникне через: ${timeLeft} секунд`;
    const timer = setInterval(() => {
        timeLeft = Math.max(0, Math.floor(duration - (new Date().getTime() -
startTime) / 1000));
        timerElement.textContent = `Цей аналіз зникне через: ${timeLeft}
секунд.`;
        if (timeLeft <= 0) {
            clearInterval(timer);
            analysisContentElement.innerHTML = '';
            timerElement.innerHTML = '';
            localStorage.removeItem('timerStart');
```

```
            // Оновлення часу закінчення аналізу в базі даних
            const userId = firebase.auth().currentUser.uid;
            const userRef = db.collection('users').doc(userId);
            userRef.get().then((doc) => {
                if (doc.exists) {
                    const userData = doc.data();
                    const latestTestResult =
userData.testsCompleted[userData.testsCompleted.length - 1];
                    latestTestResult.analysisEndTime = new Date().getTime();
                    userRef.update({ testsCompleted: userData.testsCompleted
});
                }
            });
        }
    }, 1000);
}
// Оновлення прогресу навчання
function updateLearningProgress() {
    const userId = firebase.auth().currentUser.uid;
    const userRef = db.collection('users').doc(userId);
    userRef.get().then(doc => {
        if (!doc.exists) {
            console.log("Документ користувача не знайдено!");
            return;
        }
        const userData = doc.data();
        const completedTests = userData.testsCompleted.map(test =>
test.testTitle);
        const uniqueCompletedTests = [...new Set(completedTests)]; //
Видалення дублікатів
        // Підрахунок загальної кількості доступних тестів
        db.collection("tests").get().then(querySnapshot => {
            const totalTests = querySnapshot.size;
            const progressPercentage =
Math.round((uniqueCompletedTests.length / totalTests) * 100);
            // Оновлення UI
            document.getElementById('progress-bar').style.width =
progressPercentage + '%';
            document.getElementById('progress-percentage').textContent =
progressPercentage;
        });
    }).catch(error => {
        console.log("Помилка при отриманні даних користувача:", error);
    });
}
```

src/scripts/db.js:

```
// Функція для створення або перевірки існування документа користувача в
колекції 'users'
function createUserDocument(user, callback) {
    const userRef = db.collection('users').doc(user.uid); // Отримання
посилання на документ користувача за його UID
    // Спроба отримати документ
    userRef.get().then(doc => {
        if (!doc.exists) {
            // Якщо документ не існує, створюємо його з початковими даними
            userRef.set({
                email: user.email, // Зберігаємо email користувача
                progress: 0, // Зберігаємо початковий прогрес
                testsCompleted: [], // Масив завершених тестів
                notes: [] // Масив нотаток
            }, {merge: true})
```

```
                .then(() => callback(null)) // Виклик callback-функції без
помилок
                .catch((error) => callback(error)); // Обробка можливих
помилок при збереженні
        } else {
            // Якщо документ вже існує, повертаємо успіх без створення нового
            callback(null);
        }
    }).catch((error) => {
        console.error("Error checking user document:", error); // Логування
помилок
        callback(error);
    });
}
// Функція для додавання нової задачі користувачеві
function addTaskForUser(userId, taskText, callback) {
    const userRef = db.collection('users').doc(userId); // Отримання
посилання на документ користувача
    const newTask = {text: taskText, completed: false}; // Створення нової
задачі
    // Оновлення документа з додаванням нової задачі
    userRef.update({
        notes: firebase.firestore.FieldValue.arrayUnion(newTask) // Додавання
задачі до масиву 'notes'
    })
        .then(() => callback(null)) // Успішне виконання без помилок
        .catch((error) => callback(error)); // Обробка помилок
}
// Функція для отримання всіх задач користувача
function getTasksForUser(userId, callback) {
    db.collection('users').doc(userId).get()
        .then(doc => {
            if (doc.exists) {
                callback(null, doc.data().notes); // Повернення задач, якщо
документ існує
            } else {
                callback(new Error("No such document!"), null); //
Повідомлення про відсутність документу
            }
        })
        .catch((error) => {
            callback(error, null); // Обробка помилок під час запиту
        });
}
```

src/scripts/router.js:

```
// Додаємо обробник події для завантаження вмісту сторінки
document.addEventListener('DOMContentLoaded', () => {
    loadSection('home'); // Завантажуємо головну сторінку як початковий
розділ
    initializeBreadcrumbs(); // Ініціалізуємо хлібні крихти для навігації
    // Знаходимо посилання для виходу з системи
    const logoutLink = document.querySelector('.nav-link[data-
section="logout"]');
    if (logoutLink) {
        // Додаємо обробник кліку для виходу з системи
        logoutLink.addEventListener('click', function (e) {
            e.preventDefault(); // Запобігає стандартній поведінці посилання
            firebase.auth().signOut().then(() => {
                console.log('Вихід успішний');
                window.loadSection('home'); // Завантажуємо головну сторінку
після виходу
            }).catch((error) => {
```

```
                console.error('Помилка при виході', error);
            });
        });
    }
});
// Функція для ініціалізації хлібних крихт
function initializeBreadcrumbs() {
    updateBreadcrumbs(['home']); // Створюємо початковий шлях для хлібних
крихт
}
// Функція для оновлення хлібних крихт
function updateBreadcrumbs(pathArray) {
    const breadcrumbsContainer = document.getElementById('breadcrumbs');
    breadcrumbsContainer.innerHTML = ''; // Очищаємо контейнер хлібних крихт
    // Створюємо елементи хлібних крихт з переданого масиву шляхів
    pathArray.forEach((path, index) => {
        const li = document.createElement('li');
        const a = document.createElement('a');
        a.textContent = path; // Тут можна замінити на назви, зрозумілі
користувачеві
        a.href = "#";
        a.onclick = () => {
            const sectionPath = pathArray.slice(0, index + 1).join('/');
            loadSection(sectionPath, index > 0);
        };
        li.appendChild(a);
        breadcrumbsContainer.appendChild(li);
        // Додаємо роздільник, якщо не останній елемент
        if (index < pathArray.length - 1) {
            li.appendChild(document.createTextNode(' / '));
        }
    });
}
// Функція для завантаження розділів сайту
// Функція для завантаження розділів сайту
function loadSection(sectionName, isSubsection = false) {
    const mainContent = document.querySelector('main');
    let path;
    // Визначаємо шлях до вмісту розділу
    if (sectionName === 'home') {
        path = `sections/home.html`;
    } else if (sectionName === 'login' || sectionName === 'register') {
        path = `sections/${sectionName}.html`;
    } else if (isSubsection) {
        const [section, subsection] = sectionName.split('/');
        path = `sections/${section}/${subsection}.html`;
    } else {
        path = `sections/${sectionName}/index.html`;
    }
    // Перевірка на спеціальний випадок виходу з системи
    if (sectionName === 'logout') {
        console.log('Обробка виходу...');
        return;
    }
    // Завантаження HTML вмісту за визначеним шляхом
    fetch(path)
        .then(response => response.text())
        .then(html => {
            mainContent.innerHTML = html;
            mainContent.dataset.originalContent = html;
            updateActiveLink(sectionName);
            attachCardListeners();
            // Розділення секції на масив шляхів для хлібних крихт
            const paths = sectionName.split('/');
```

```
            updateBreadcrumbs(paths);
            if (sectionName === 'dashboard') {
                updateUserInfo();
                setupAddTaskButton();
                displayTasks();
                loadAvailableTests();
                displayTestHistory();
                displayErrorAnalysis();
                setupThemeToggle();  // Встановлення перемикача теми
            }
        })
        .catch(error => console.error('Error loading section:', error));
}
function setupThemeToggle() {
    var toggleButton = document.getElementById('theme-toggle');
    if (toggleButton) {
        toggleButton.addEventListener('click', function() {
            document.body.classList.toggle('dark-theme');
            const theme = document.body.classList.contains('dark-theme') ?
'dark' : 'light';
            localStorage.setItem('theme', theme);
        });
        // Перевірка теми при завантаженні
        if (localStorage.getItem('theme') === 'dark') {
            document.body.classList.add('dark-theme');
        }
    } else {
        console.error('Element not found!');
    }
}
// Функція для оновлення активних посилань у навігації
function updateActiveLink(sectionName) {
    // Видаляємо клас 'active' з усіх посилань
    document.querySelectorAll('.nav-link').forEach(link => {
        link.classList.remove('active'); // Видалення класу 'active' з усіх
посилань
    });
    // Додавання класу 'active' до активного посилання
    document.querySelectorAll(`.nav-link[data-section="${sectionName}"],
.nav-link[data-main-section="${sectionName}"]`).forEach(link => {
        link.classList.add('active');
    });
}
// Функція для навішування обробників подій на навігаційні посилання
document.querySelectorAll('.nav-link').forEach(link => {
    link.addEventListener('click', (e) => {
        e.preventDefault(); // Запобігання стандартній поведінці браузера при
кліку
        const sectionName = e.target.getAttribute('data-section'); //
Отримання назви розділу з атрибуту посилання
        loadSection(sectionName); // Завантаження відповідного розділу
    });
});
// Функція для навішування обробників подій на картки
function attachCardListeners() {
    document.querySelectorAll('.card').forEach(card => {
        // При кліку на картку
        card.addEventListener('click', () => {
            const section = card.getAttribute('data-section'); // Отримання
секції з атрибуту картки
            const mainSection = card.getAttribute('data-main-section'); //
Отримання основної секції з атрибуту картки
            const isSubsection = section !== mainSection; // Перевірка, чи є
поточна секція підрозділом
```

```
            loadSection(`${mainSection}/${section}`, isSubsection); // 
Завантаження секції
        });
    });
}
// Обробник змін стану автентифікації
firebase.auth().onAuthStateChanged(function (user) {
    if (user) {
        // Користувач увійшов
        document.querySelector('.nav-link[data-
section="dashboard"]').style.display = ''; // Показувати посилання на панель 
керування
        document.querySelector('.nav-link[data-
section="login"]').style.display = 'none'; // Сховати посилання на вхід
        document.querySelector('.nav-link[data-
section="register"]').style.display = 'none'; // Сховати посилання на 
реєстрацію
        document.querySelector('.nav-link[data-
section="logout"]').style.display = ''; // Показувати посилання на вихід
    } else {
        // Користувач вийшов
        document.querySelector('.nav-link[data-
section="dashboard"]').style.display = 'none'; // Сховати панель керування
        document.querySelector('.nav-link[data-
section="login"]').style.display = ''; // Показувати посилання на вхід
        document.querySelector('.nav-link[data-
section="register"]').style.display = ''; // Показувати посилання на 
реєстрацію
        document.querySelector('.nav-link[data-
section="logout"]').style.display = 'none'; // Сховати посилання на вихід
    }
});
// Функція для оновлення інформації користувача
function updateUserInfo() {
    const user = firebase.auth().currentUser;
    if (user) {
        const userRef = db.collection('users').doc(user.uid);
        userRef.get().then(doc => {
            if (doc.exists) {
                const userData = doc.data();
                const userEmailEl = document.getElementById('user-email');
                const userNameEl = document.getElementById('first-name');
                const userLastNameEl = document.getElementById('lastname');
                if (userEmailEl) userEmailEl.textContent = user.email;  // 
Оновлення електронної пошти користувача
                if (userNameEl) userNameEl.textContent = userData.firstName; 
// Оновлення імені користувача
                if (userLastNameEl) userLastNameEl.textContent = 
userData.lastName; // Оновлення прізвища користувача
            } else {
                console.log("No user data available."); // Логування 
відсутності даних користувача
            }
        }).catch(error => {
            console.error("Error fetching user data:", error); // Логування 
помилок при отриманні даних
        });
    }
}
```

src/index.html:

```
<!DOCTYPE html>
<html lang="uk"> <!-- Відкриття тега html з атрибутом мови -->
<head> <!-- Відкриття тега head, який містить метадані про документ -->
```

```
    <meta charset="UTF-8"> <!-- Встановлення кодування документа -->
    <meta http-equiv="Content-Security-Policy" content="script-src 'self'
'unsafe-inline' https://www.gstatic.com; frame-src 'self'
https://accounts.google.com;">
    <!-- Встановлення політики безпеки контенту -->
    <title>Electron Guide</title> <!-- Заголовок вебсторінки -->
    <link rel="stylesheet" type="text/css" href="styles/main.css"> <!--
Підключення CSS-файлу -->
    <!-- Підключення шрифтів з Google Fonts -->
    <link rel="preconnect" href="https://fonts.googleapis.com">
    <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
    <link
href="https://fonts.googleapis.com/css2?family=Comfortaa:wght@300..700&family
=Pacifico&display=swap"
          rel="stylesheet">
    <link
href="https://fonts.googleapis.com/css2?family=Montserrat:ital,wght@0,100..90
0;1,100..900&display=swap"
          rel="stylesheet">
</head>
<body> <!-- Відкриття тега body, який містить вміст веб-сторінки -->
<div class="main-container"> <!-- Головний контейнер для вмісту сторінки -->
    <div class="sidebar"> <!-- Бічна панель навігації -->
        <nav>  <!-- Тег для навігації -->
            <ul class="top-links"> <!-- Ненумерований список для навігаційних
посилань -->
                <!-- Кожен елемент списку містить посилання на різні розділи
сайту -->
                <li><a class="nav-link" data-section="home" data-main-
section="home">
                    <svg class="home-icon" width="20" height="19" viewBox="0
0 20 19" fill="none"
                         xmlns="http://www.w3.org/2000/svg">
                        <path d="M3.25513 7.07788L10.3384 1.5687L17.4216
7.07788V15.7352C17.4216 16.1526 17.2558 16.553 16.9606 16.8482C16.6654
17.1434 16.265 17.3092 15.8476 17.3092H4.82918C4.41172 17.3092 4.01135
17.1434 3.71616 16.8482C3.42096 16.553 3.25513 16.1526 3.25513
15.7352V7.07788Z"
                              fill="white" fill-opacity="0.18"/>
                        <path d="M7.97729 17.3092V9.43897H12.6995V17.3092"
fill="white" fill-opacity="0.18"/>
                        <path d="M3.25513 7.07788L2.84097 6.54539C2.67664
6.6732 2.58053 6.86971 2.58053 7.07788H3.25513ZM10.3384 1.5687L10.7525
1.0362C10.5089 0.846733 10.1678 0.846733 9.92421 1.0362L10.3384
1.5687ZM17.4216 7.07788H18.0962C18.0962 6.86971 18.0001 6.6732 17.8358
6.54539L17.4216 7.07788ZM3.25513 15.7352H2.58053H3.25513ZM7.30269
17.3092C7.30269 17.6818 7.60472 17.9838 7.97729 17.9838C8.34986 17.9838
8.65188 17.6818 8.65188 17.3092H7.30269ZM7.97729 9.43897V8.76437C7.60472
8.76437 7.30269 9.0664 7.30269 9.43897H7.97729ZM12.6995 9.43897H13.374C13.374
9.0664 13.072 8.76437 12.6995 8.76437V9.43897ZM12.0249 17.3092C12.0249
17.6818 12.3269 17.9838 12.6995 17.9838C13.072 17.9838 13.374 17.6818 13.374
17.3092H12.0249ZM3.66929 7.61038L10.7525 2.10119L9.92421 1.0362L2.84097
6.54539L3.66929 7.61038ZM9.92421 2.10119L17.0075 7.61038L17.8358
6.54539L10.7525 1.0362L9.92421 2.10119ZM16.747
7.07788V15.7352H18.0962V7.07788H16.747ZM16.747 15.7352C16.747 15.9737 16.6523
16.2025 16.4836 16.3712L17.4376 17.3252C17.8593 16.9035 18.0962 16.3316
18.0962 15.7352H16.747ZM16.4836 16.3712C16.3149 16.5399 16.0861 16.6346
15.8476 16.6346V17.9838C16.4439 17.9838 17.0159 17.7469 17.4376
17.3252L16.4836 16.3712ZM15.8476
16.6346H4.82918V17.9838H15.8476V16.6346ZM4.82918 16.6346C4.59063 16.6346
4.36185 16.5399 4.19317 16.3712L3.23915 17.3252C3.66085 17.7469 4.2328
17.9838 4.82918 17.9838V16.6346ZM4.19317 16.3712C4.02449 16.2025 3.92972
15.9737 3.92972 15.7352H2.58053C2.58053 16.3316 2.81744 16.9035 3.23915
17.3252L4.19317 16.3712ZM3.92972
```

```
15.7352V7.07788H2.58053V15.7352H3.92972ZM8.65188
17.3092V9.43897H7.30269V17.3092H8.65188ZM7.97729
10.1136H12.6995V8.76437H7.97729V10.1136ZM12.0249
9.43897V17.3092H13.374V9.43897H12.0249Z"
                                fill="#E2F1FF"/>
                    </svg>
                    Головна</a></li>
                <li><a class="nav-link" data-section="html-basics" data-main-
section="html-basics">HTML</a></li>
                <li><a class="nav-link" data-section="css-basics" data-main-
section="css-basics">CSS</a></li>
                <li><a class="nav-link" data-section="js-basics" data-main-
section="js-basics">JavaScript</a></li>
                <li><a class="nav-link" data-section="frameworks-and-
libraries"
                        data-main-section="frameworks-and-
libraries">Фреймворки</a></li>
                <li><a class="nav-link" data-section="server-side-dev" data-
main-section="server-side-dev">Back-end</a>
                </li>
                <li><a class="nav-link" data-section="web-api" data-main-
section="web-api">Web API</a></li>
                <li><a class="nav-link" data-section="web-app-security" data-
main-section="web-app-security">Безпека</a>
                </li>
                <li><a class="nav-link" data-section="deployment-and-hosting"
                        data-main-section="deployment-and-
hosting">Розгортання</a></li>
            </ul>
            <ul class="bottom-links">
                <li><a class="nav-link" data-section="dashboard" data-main-
section="dashboard">Профіль</a></li>
                <li><a class="nav-link" data-section="login" data-main-
section="login">Вхід</a></li>
                <li><a class="nav-link" data-section="register" data-main-
section="register">Реєстрація</a></li>
                <li><a class="nav-link" data-section="logout" data-main-
section="logout">Вихід</a></li>
            </ul>
        </nav>
    </div>
    <div class="content"> <!-- Контейнер для основного вмісту сторінки -->
        <!-- Це навігація по підрозділах -->
        <div class="breadcrumbs-container">
            <ul id="breadcrumbs">
                <!-- Тут будуть breadcrumbs -->
            </ul>
        </div>
        <main id="main-content">
            <!-- Тут буде основний контент -->
        </main>
    </div>
</div>
<footer>
</footer>
</body>
<!-- Підключення скриптів Firebase -->
<script src="https://www.gstatic.com/firebasejs/8.0.0/firebase-
app.js"></script>
<script src="https://www.gstatic.com/firebasejs/8.0.0/firebase-
auth.js"></script>
<script src="https://www.gstatic.com/firebasejs/8.0.0/firebase-
firestore.js"></script>
<script>
```

```
    // Your web app's Firebase configuration
    const firebaseConfig = {
        apiKey: process.env.API_KEY,
        authDomain: process.env.AUTH_DOMAIN,
        projectId: process.env.PROJECT_ID,
        storageBucket: "web-apps-manual.appspot.com",
        messagingSenderId: process.env.MESSAGING_SENDER_ID,
        appId: process.env.APP_ID
    };
    // Initialize Firebase
    firebase.initializeApp(firebaseConfig);
    const db = firebase.firestore();
</script>
<!-- Підключення локальних JavaScript-файлів -->
<script src="scripts/dashboard.js"></script>
<script src="scripts/db.js"></script>
<script src="scripts/router.js"></script>
<script src="scripts/auth.js"></script>
</html>
```

src/sections/home.html:

```
<section id="home">
    <h2>Перший крок до програмування застосунків!</h2>
    <div class="text-p">
        <p>Цей посібник розроблено з метою надати початківцям усі необхідні
знання та інструменти для створення сучасних
            вебзастосунків. Ви пройдете через всі етапи розробки
вебзастосунків — від основ HTML, CSS та JavaScript до
            використання передових фреймворків і технологій.</p>
    </div>
    <h3>Огляд вебтехнологій</h3>
    <div class="list">
        <div class="item">
            <p>HTML (HyperText Markup Language): Мова розмітки, яка
використовується для створення структури
                веб-сторінок.</p>
        </div>
        <div class="item">
            <p>CSS (Cascading Style Sheets): Мова стилів, що використовується
для задання вигляду веб-сторінок.</p>
        </div>
        <div class="item">
            <p>JavaScript: Програмна мова, що дозволяє робити вебсторінки
інтерактивними та динамічними.</p>
        </div>
        <div class="item">
            <p>Фреймворки і бібліотеки: Інструменти, які спрощують розробку,
надаючи вже готові до використання
                компоненти та функції (наприклад, React, Vue.js для фронтенду
та Node.js для бекенду).</p>
        </div>
        <div class="item">
            <p>Бази даних: Системи для зберігання та управління даними
(наприклад, MongoDB, PostgreSQL).</p>
        </div>
    </div>
    <div class="get-started">
        <h3>Як почати?</h3>
        <p>Оберіть розділ, який вас цікавить, з меню навігації, або
використайте особистий кабінет для відстеження
            прогресу та проходження тестування.</p>
    </div>
</section>
```

Алгоритми

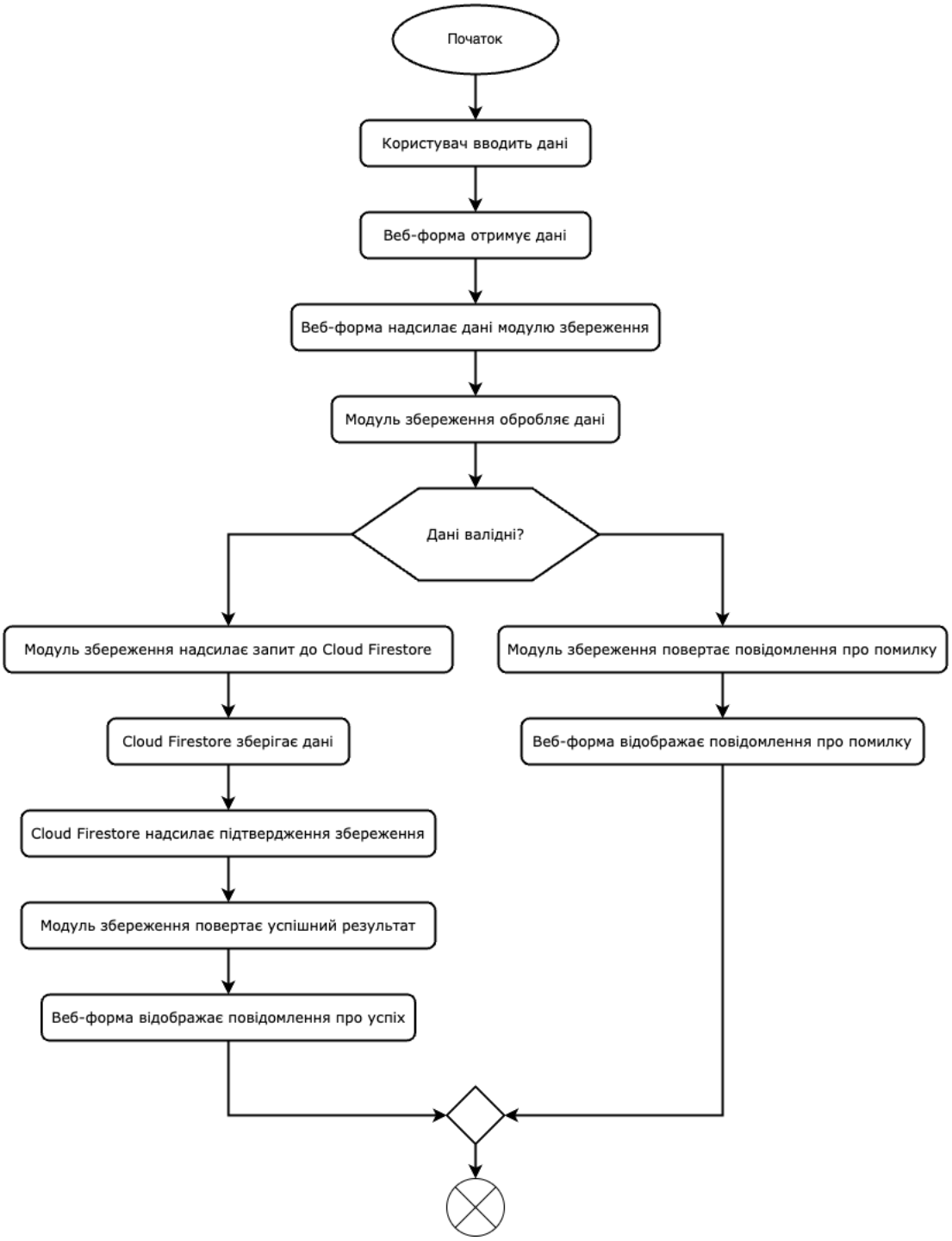


Рис. В.1. Алгоритм збереження даних

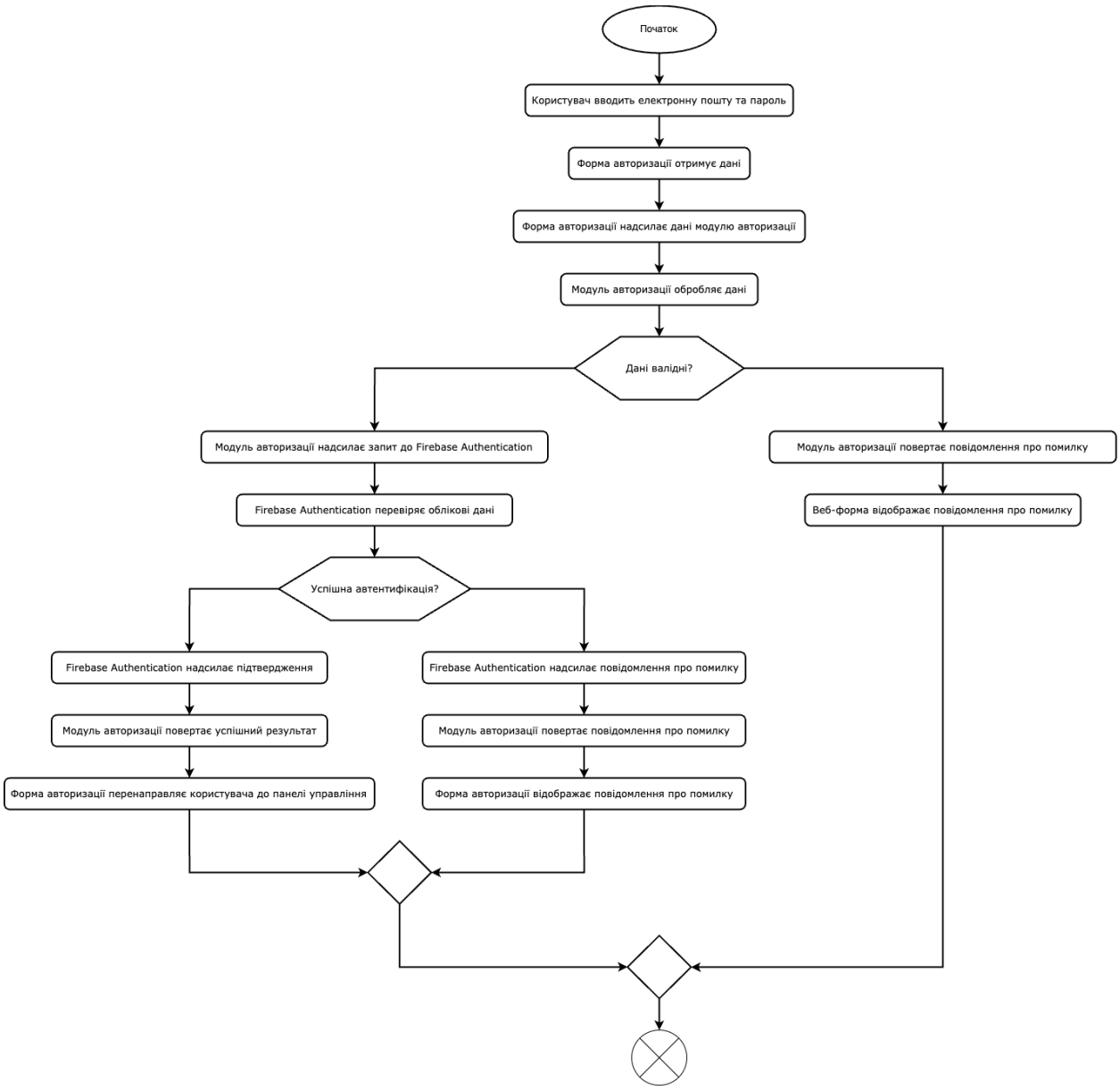


Рис. В.2. Алгоритм авторизації користувача

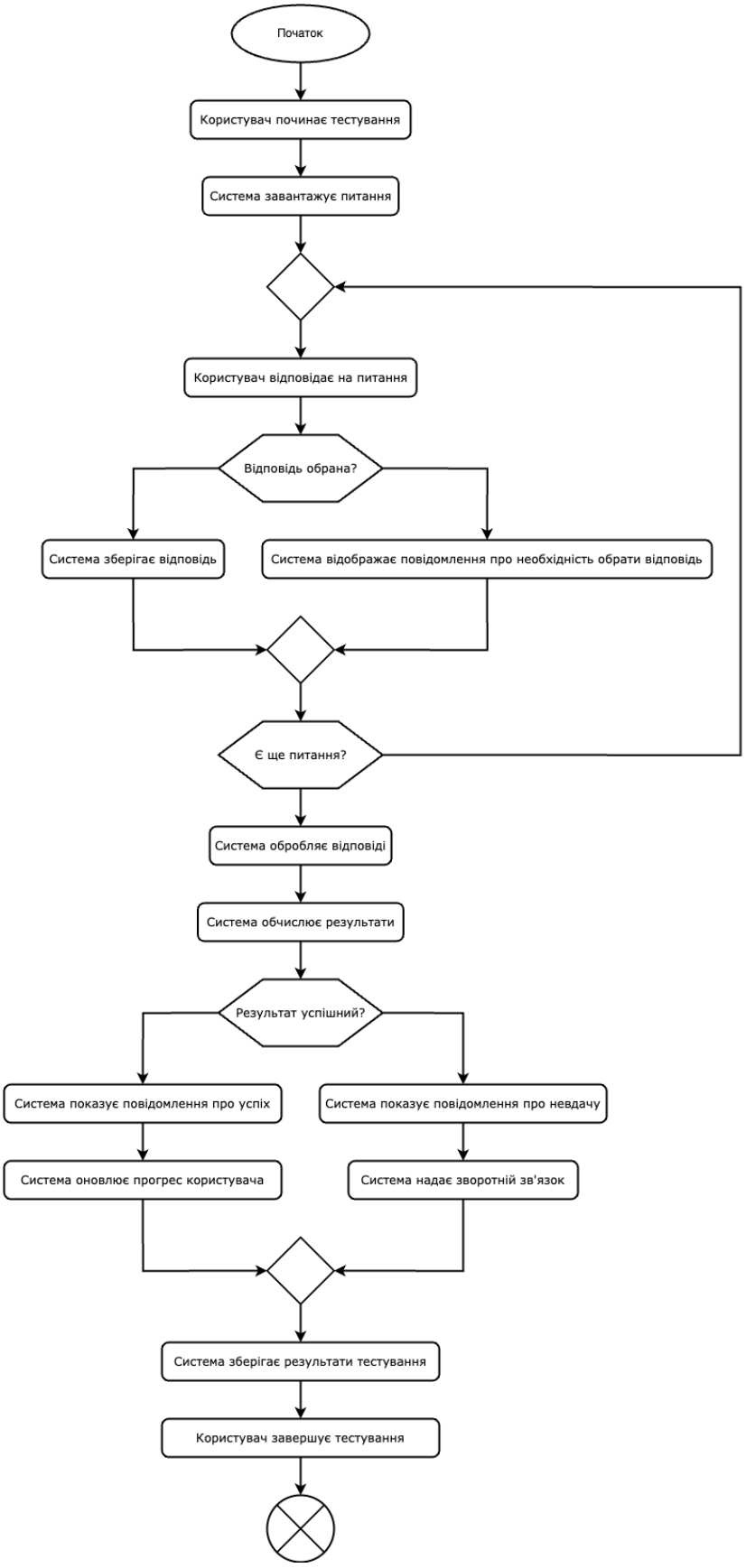


Рис. В.3. Алгоритм тестування знань