МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ОДЕСЬКИЙ ДЕРЖАВНИЙ ЕКОЛОГІЧНИЙ УНІВЕРСИТЕТ

Факультет комп'ютерних наук,
управління та адміністрування
Кафедра інформаційних технологій

Кваліфікаційна робота бакалавра

на тему: Реалізація десктопного застосунку щодо організації робочого простору студента

Виконав студент групи **КН-20**
спеціальності 122 Комп'ютерні науки
Коваленко Владислав Сергійович

Керівник **асистент**
Клепатська В.В.

Консультант **д.т.н., професор**
Казакова Н.Ф.

Рецензент **заступник директора**
- начальник відділу Департаменту
інформації та цифрових рішень
Одеської міської ради
Корчемний П.А.

Одеса 2023

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ І ТЕРМІНІВ

Програмне забезпечення – сукупність програм системи оброблення інформації та програмних документів, необхідних для експлуатації цих програм.

CSS – це спеціальна мова (мова стилів), за допомогою якої описують вигляду документів (як і де відображати елементи веб-сторінки), написаних мовами розмітки даних.

Desktop застосунки – це програми, які потребують ОС настільного комп'ютера для своєї роботи.

HTML – стандартизована мова розмітки документів для перегляду вебсторінок у браузері.

Integrated development environment або англ. IDE) – комплексне програмне рішення для розробки програмного забезпечення.

JSON (JavaScript Object Notation) – це текстовий формат, призначений для зберігання структурованих даних.

npm (Node Package Manager) – це менеджер пакунків для мови програмування JavaScript.

ПЗ	– Програмне Забезпечення.
CSS	– Cascading Style Sheets.
HTML	– HyperText Markup Language.
JSON	– JavaScript Object Notation.
npm	– Node package manager.
TDL	– To Do List.
UI	– User Interface.
VSCode	– Visual Studio Code.
IDE	– Integrated Development Environment.

ВСТУП

Задача розробки десктопного застосунку "Організація робочого простору студента" передбачає вирішення ряду проблем, з якими стикаються студенти під час організації свого навчального процесу. Основна мета цього проєкту – створення зручного, ефективного та функціонального інструменту, який допоможе студентам у кращій організації їхнього робочого простору та забезпечить їм ефективну навчальну діяльність.

Застосунок має надавати можливість студентам швидко та зручно переглядати розклад пар. Він повинен бути легким у використанні та інтуїтивно зрозумілим, щоб студенти могли швидко знайти необхідну інформацію про свої пари, таку як час початку та закінчення, місце проведення, назва предмета та викладачі.

Одним з головних функціональних елементів застосунку є можливість швидкого підключення до пари. Це дозволить студентам без зайвих зусиль приєднатися до віртуальних аудиторій або платформ для онлайн-навчання. Застосунок повинен забезпечити зручний доступ до посилань та вмісту, пов'язаного з кожною парою, та забезпечити можливість автоматичного переходу до платформи навчання з одного кліку.

Мета кваліфікаційної роботи бакалавра полягає у створенні інструменту, який сприятиме покращенню організації навчального процесу студентів. Даний застосунок повинен бути зручним, ефективним та легко розширюваним, щоб задовольнити потреби студентів та стати незамінним помічником у їхній навчальній діяльності.

Робочий простір студента не обмежується лише розкладом пар. Застосунок також має включати функціонал To-Do List, який дозволить студентам створювати, відстежувати та керувати своїми завданнями. Вони зможуть додавати нові завдання, встановлювати терміни виконання, позначати завдання як виконані та видаляти їх. Це допоможе студентам більш ефективно організувати свої студентські обов'язки та забезпечить їм систематичний підхід до

виконання завдань.

Однією з ключових цілей проєкту є надання можливості іншим студентам доповнювати та вносити зміни до застосунку завдяки GitHub. Тому весь код проєкту буде відкритим та супроводжуватиметься коментарями, що роз'яснюють його структуру, функціональність та логіку. Це дозволить іншим студентам легко орієнтуватися в коді, зрозуміти його принципи та вносити свої власні покращення до проєкту.

Дана кваліфікаційна робота бакалавра складається з 66 сторінок, 23 рисунків та 9 джерел посилання.

1. ПОРІВНЯЛЬНИЙ АНАЛІЗ СУЧАСНИХ СИСТЕМ, ЗАСТОСУНКІВ ДЛЯ ОРГАНІЗАЦІЇ РОБОЧОГО ПРОСТОРУ СТУДЕНТА

На сьогоднішній день існує декілька застосунків та платформ, які допомагають студентам організувати свій робочий простір та розклад занять. Для проведення аналізу, було проаналізовано деякі з найпоширеніших та популярних систем, що пропонують подібні функціональності:

- Google Calendar;
- Microsoft Outlook;
- Todois;
- Notion.

Застосунок повинен відповідати наступним вимогам:

- спеціалізований робочий простір;
- персоналізація;
- удосконалені нагадування;
- інтеграція з іншими функціями;
- локальне збереження та безпека даних.

Застосунок, повинен бути зосередженим на потребах студента і надає спеціалізований робочий простір, спрямований на управління розкладом занять, завданнями та іншими важливими елементами, які студентам необхідні для організації свого навчання. Застосунок повинен дозволяти студентам налаштувати розклад занять та завдання відповідно до їхніх особистих потреб. Розробник повинен надати можливість налаштування пріоритетів, категорій, тегів, кольорових позначок і т.д., що допоможе студентам краще організувати свій навчальний процес. Застосунок повинен пропонувати розширені функції нагадувань, які враховують специфічні потреби студента. Наприклад, можливість встановлення нагадувань про терміни здачі завдань, важливі події або навіть рекомендації щодо часу навчання та відпочинку. Застосунок повинен поєднувати у собі різноманітні функції, які спрощують роботу студента. На-

приклад, інтеграція з системою управління завданнями, можливість збереження нотаток або додавання матеріалів для вивчення безпосередньо у розклад занять. Застосунок, який працює на пристрої, повинен забезпечити локальне збереження даних, що дозволить студентам мати повний контроль над своїми особистими даними та безпекою. Це особливо важливо, коли йдеться про приватну інформацію, таку як розклад навчання. Загалом, застосунок повинен мати переваги, орієнтовані на специфічні потреби студентів, надаючи зручний та персоналізований інструмент для організації робочого простору та навчального процесу.

1.1 Опис застосунку Google Calendar

Google Calendar є однією з найпопулярніших онлайн-календарних систем, що надає широкий набір функцій для організації розкладу, нагадувань та співпраці з іншими користувачами. Однак, застосунок має переваги порівняно з Google Calendar в контексті організації робочого простору студента (рис.1).

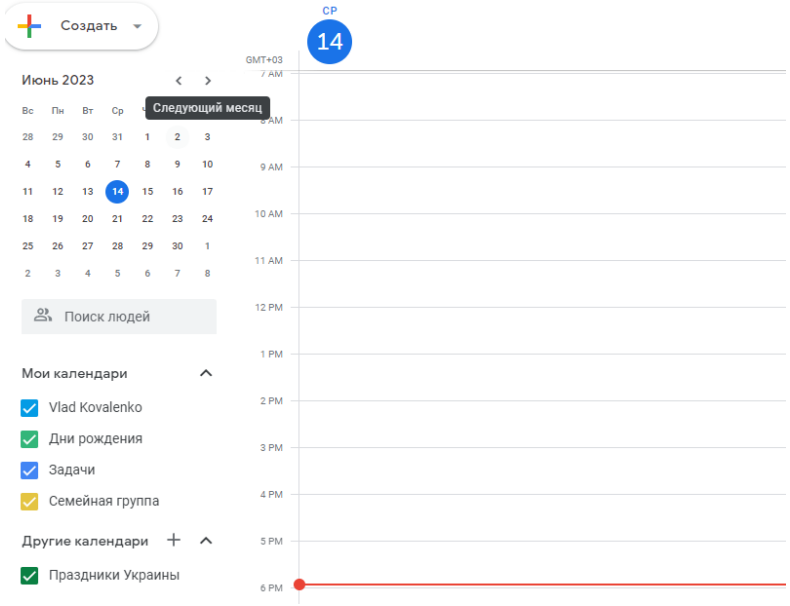

Рисунок 1 – Інтерфейс Google Calendar

1.2 Опис застосунку Microsoft Outlook

Microsoft Outlook є популярним клієнтом електронної пошти та календарем. Він має функціональність для створення та керування розкладом пар, встановлення нагадувань та спільної роботи з іншими користувачами. Проте, Microsoft Outlook зазвичай використовується для робочих потреб і не має спеціалізованих функцій для організації робочого простору студента (див.рис. 2).

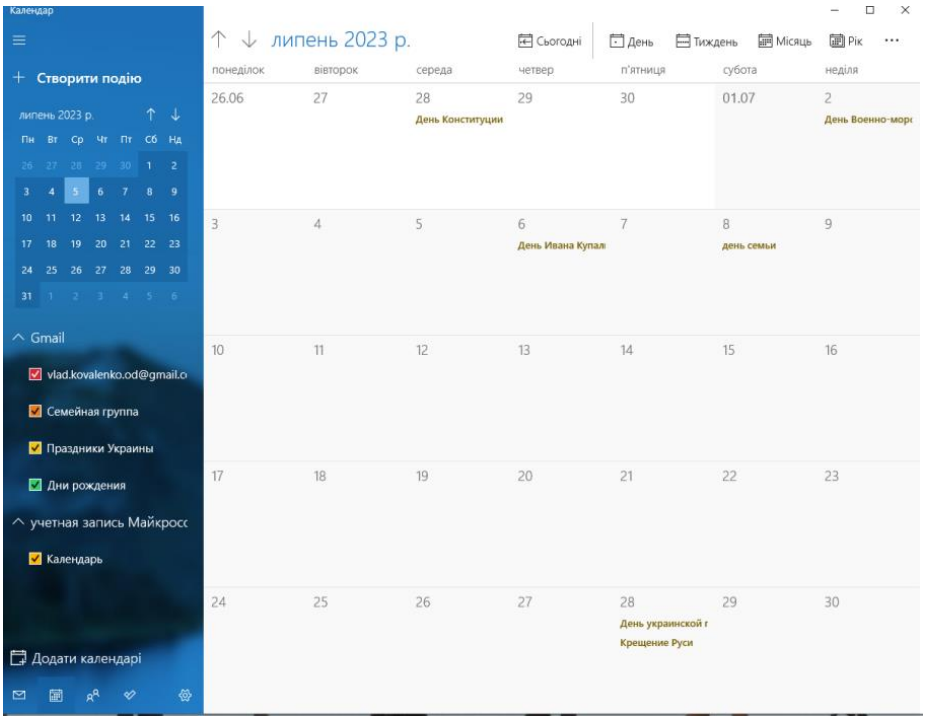

Рисунок 2 – Інтерфейс Microsoft Outlook

1.3 Опис застосунку Todois

Todoist є популярним застосунком для створення та керування списками завдань. Він дозволяє студентам створювати та організовувати свої завдання за пріоритетами, встановлювати дедлайни та відстежувати прогрес виконання завдань. Однак, Todoist не має спеціалізованих функцій для організації розкладу пар та інтеграції з календарями (рис.3).

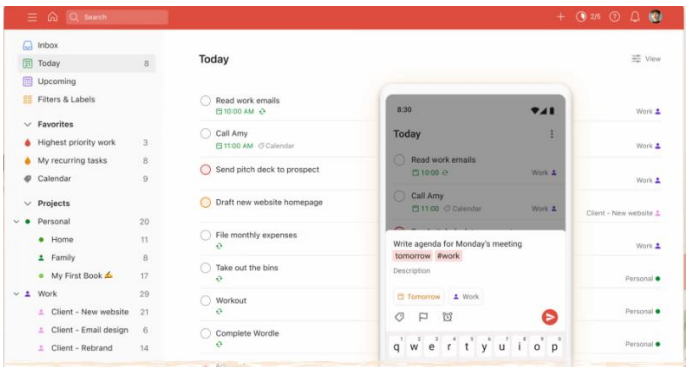

Рисунок 3 – Інтерфейс Todois

1.4 Опис застосунку Notion

Notion є комплексним інструментом для організації робочого простору. Основним недоліком програми є перевантаженість та закритий код, який не дозволяє додати студентам комп'ютерних технологій свій функціонал (рис.4).

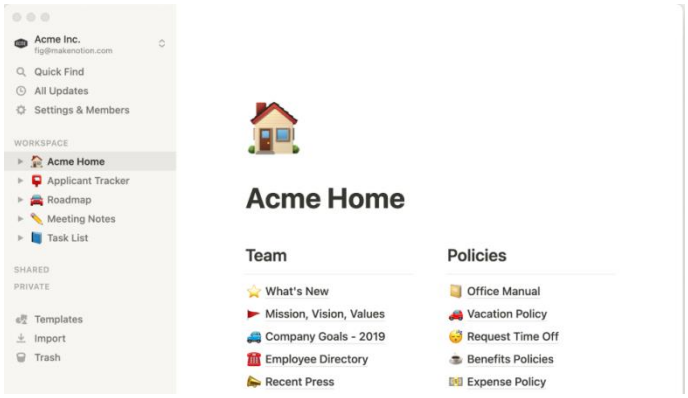

Рисунок 4 – Інтерфейс Notion

1.5 Обґрунтування необхідності та актуальності розробки нового застосунку розробки

Переглянувши усі аналоги на ринку, можна зробити висновки, що жоден не зібрав у собі ті риси, які дозволили б студенту організувати робочий простір.

Метою розробки застосунку для організації робочого просто студента є:

- набуття практичних навичок у побудові архітектури, та розробки програмного забезпечення, використання сучасних фреймворків на мові програмування JavaScript;
- створення власного програмного продукту засобами функціонально – орієнтованими середовища мови програмування JavaScript, та дослідження його складових компонентів.

Основні цілі, які можна поставити для виконання проєкту:

- зацікавити студентів до використання застосунка;
- зробити навчання зручнішим;
- зібрати спілку ініціативних студентів;
- збільшити загальний рівень відвідування пар;
- збільшити загальний рівень навчальної успішності студентів.

1.6 Вимоги до функціональних характеристик застосунку

Вимоги – це можливість швидко зорієнтуватися у розкладі, записати задачі та реалізувати доповнення застосунку. Програмний продукт повинен володіти наступними функціональними характеристиками:

- мати можливість до подальшого використання;
- мати оптимальну функціональніть;
- мати відкритий код;
- мати зрозумілу архітектуру;
- бути реалізованим на популярних та сучасних технологіях;
- мати високий рівень швидкості;
- знаходитися у середовищі з достатнім функціоналом для підтримки його іншими розробниками.

2. ВИБІР ПРОГРАМНИХ ЗАСОБІВ ДЛЯ РЕАЛІЗАЦІЇ ДЕСКТОПНОГО ЗАСТОСУНКУ

2.1 Обґрунтування вибору мов програмування для розробки десктопного застосунку

Існує багато мов програмування, кожна з яких має свою власну історію та призначення. Нижче представлено кілька популярних мов програмування та короткий опис кожної з них [1]:

– C;
– Java;
– Python.

Мова програмування C була створена у 1972 році Денісом Рітчі в Bell Laboratories. Вона є однією з найважливіших мов програмування і знаходить широке застосування в розробці операційних систем, вбудованих систем, компіляторів та інших критичних застосунків. C відома своєю ефективністю, низькорівневим доступом до апаратного забезпечення та великою кількістю бібліотек. Розробка мови Java розпочалась у 1991 році у Sun Microsystems (пізніше придбаному компанією Oracle). Вона була створена Джеймсом Гослінгом та його командою з метою створення мови програмування для розробки надійних та переносних програм для вбудованих систем. Java отримала широке поширення завдяки своїй платформонезалежності, об'єктно-орієнтованому підходу та великій кількості доступних бібліотек. Python був розроблений у 1991 році Гвідо ван Россумом. Він був задуманий як проста для вивчення мова програмування, що поєднує лаконічний синтаксис з потужними можливостями. Python широко використовується у різних галузях, включаючи веб-розробку, наукові обчислення, штучний інтелект та автоматизацію завдань. Він відомий своєю простотою, зрозумілістю коду та багатством сторонніх бібліотек.

Це лише кілька прикладів мов програмування, але існує безліч інших, таких як C++, C#, Ruby, Swift, Go тощо, кожна з яких має свої особливості та сфери застосування.

Мови програмування можна поділити на різні групи залежно від різних критеріїв. Нижче представлено кілька основних способів класифікації мов програмування:

– низькорівневі та високорівневі мови;
– процедурні та об'єктно-орієнтовані мови;
– компільовані та інтерпретовані мови;
– функціональні мови.

Класифікація низькорівневі та високорівневі мови базується на рівні абстракції, яку надає мова програмування. Низькорівневі мови, такі як мова асемблера, надають прямий доступ до апаратного забезпечення комп'ютера та мають високу рівень контролю над ним. Високорівневі мови, такі як Python або Java, надають більш високий рівень абстракції та простоти використання, орієнтуючись на завдання програміста, а не безпосередньо на роботу з апаратурою.

Поділ на процедурні та об'єктно-орієнтовані мови відображає підхід до організації та структури програм. Процедурні мови, такі як C або Pascal, базуються на послідовності виконання функцій та процедур. Об'єктно-орієнтовані мови, такі як Java або C++, спрямовані на організацію програм навколо об'єктів, які поєднують дані та функції, що з ними пов'язані.

Така класифікація, як компільовані та інтерпретовані мови відноситься до способу виконання програм. Компільовані мови, такі як C++ або Go, перетворюють вихідний код програми на машинний код, що може бути виконаний безпосередньо на комп'ютері. Інтерпретовані мови, такі як Python або JavaScript, використовують спеціальну програму, інтерпретатор, для виконання програм шляхом поетапного виконання їх інструкцій.

Функціональні мови – ця група мов програмування покладає акцент на функції як основний будівельний блок програм. Функціональні мови, такі як Haskell або Lisp, стимулюють використання математичних функцій, безпідказних обчислень та імутабельних даних.

JavaScript є високорівневою, об'єктно-орієнтованою мовою програмування, спеціально розробленою для використання у веб-браузерах. Вона надає можливість створювати динамічні веб-сторінки та взаємодіяти з користувачем. JavaScript володіє такими особливостями: динамічна типізація, функціональні можливості та обробка подій.

Змінні в JavaScript можуть змінювати свій тип під час виконання програми, що надає гнучкості при роботі з даними. JavaScript підтримує функції в якості об'єктів першого класу, що дозволяє передавати їх як аргументи, повертати з функцій та зберігати в змінних. JavaScript є основним механізмом обробки подій у веб-браузерах. Він дозволяє реагувати на дії користувача та взаємодіяти зі структурою веб-сторінки.

JavaScript також має велику екосистему фреймворків та бібліотек, які допомагають розробникам упростити та прискорити процес розробки веб-застосунку. Найпопулярніші фреймворки JavaScript включають: React, Angular, Vue. JavaScript став широко використовуваним не тільки на клієнтській, але й на серверній частині веб-застосунків завдяки платформі Node.js. Це дозволяє розробникам використовувати JavaScript для написання серверного коду та побудови повноцінних веб-застосунків. Загалом, JavaScript є потужною та універсальною мовою програмування, яка дозволяє розробникам створювати веб-застосунки з багатим функціоналом та взаємодією з користувачем. Вона продовжує розвиватись та займає важливе місце у веб-розробці.

Фреймворк React – це потужний фреймворк для розробки інтерфейсу користувача (UI), який був розроблений компанією Facebook. Він використовується для побудови веб-застосунків з високою продуктивністю, масштабованістю та модульністю. Основна ідея React полягає у розбитті інтерфейсу на компоненти, які можуть бути перевикористовуваними та оновлюватися незалежно один від одного. Основні принципи та особливості React включають:

- компонентна структура;
- віртуальний DOM;
- односторонній потік даних;

– JSX;
– підтримка реактивного програмування.

React дозволяє розбити інтерфейс на невеликі незалежні компоненти, які можуть мати свою внутрішню логіку та стан. Це полегшує розробку, тестування та управління складними інтерфейсами. React використовує віртуальний DOM (Document Object Model) для ефективної оновлення інтерфейсу. Замість безпосереднього маніпулювання реальним DOM, React порівнює віртуальний DOM зі змінами та виконує лише необхідні оновлення, що забезпечує високу продуктивність застосунків. React використовує односторонній потік даних, де дані витікають вниз по ієрархії компонентів. Це полегшує відслідковування та керування станом застосунку.÷ JSX є розширенням синтаксису JavaScript, яке дозволяє писати HTML-подібний код прямо в JavaScript файлі. Це полегшує створення компонентів та їх розмітку. React має ряд інструментів для реактивного програмування, таких як "хуки" (hooks), які дозволяють зручно працювати зі станом та ефектами.

React [2] також має велику екосистему розширень та бібліотек, які допомагають розробникам розширити можливості фреймворка. Найпопулярніші з них включають:

– Redux: бібліотека для керування станом застосунків React (вона допомагає організувати та управляти станом застосунків в одному місці);
– React Router: бібліотека для маршрутизації в React-застосунках (вона дозволяє створювати багатосторінкові застосунки з різними шляхами (URL) та переходами між ними);
– Material-UI: бібліотека компонентів, яка надає готові елементи дизайну в стилі Material Design (вона дозволяє швидко створювати стильні та сучасні інтерфейси).

React є одним з найпопулярніших фреймворків для розробки веб-застосунків і знаходить широке застосування у веб-розробці. Його активна спіль-

нота та постійні оновлення роблять його потужним інструментом для створення високоякісних та інтерактивних застосунків.

Node.js (або просто Node) [3] є виконавчим середовищем, побудованим на базі двигуна V8 JavaScript, який використовується для розробки серверних та мережевих застосунків. Основна перевага Node полягає в тому, що він дозволяє виконувати JavaScript на стороні сервера, що робить його універсальним для розробки як фронтенду, так і бекенду застосунків.

Основні особливості Node: платформа-сервер, подієва модель, пакетний менеджер npm та розширення функціональності.

Node надає зручну платформу для розробки серверних застосунків. Він дозволяє обробляти запити, керувати файлами, взаємодіяти з базами даних та іншими зовнішніми ресурсами. Node працює за подієвою моделлю, що означає, що він виконує запити асинхронно та не блокує виконання інших операцій. Це дозволяє створювати швидкі та ефективні застосунки, які можуть обробляти багато запитів одночаснonpm (Node Package Manager) є стандартним пакетним менеджером для Node. Він дозволяє легко встановлювати, оновлювати та керувати залежностями проєкту. За допомогою npm може швидко додавати сторонні пакети та модулі до свого проєкту. Node дозволяє розширити функціональність за допомогою модулів. Існує велика кількість модулів, доступних через npm, які надають різноманітні можливості, включаючи роботу з базами даних, обробку зображень, роботу з мережею та багато іншого.

Node Package Manager (npm) є пакетним менеджером, який входить до складу Node. Він дозволяє розробникам легко встановлювати, оновлювати та керувати залежностями своїх проєктів. Основні можливості та використання npm: установка пакетів, керування залежностями, публікація пакетів, сценарії та версіонування.

За допомогою команди npm install, можна встановити необхідні пакети для проєкту. npm автоматично завантажує пакети з репозиторію npm та їх залежностінpm дозволяє керувати залежностями проєкту. Залежності визнача-

ються у файлі package.json, де вказується список пакетів, необхідних для проєкту, а також їх версії. Завдяки цьому, команда npm install встановлює правильні версії пакетів з урахуванням залежностей. Розробник також може публікувати власні пакети в репозиторії npm, щоб інші розробники могли використовувати їх у своїх проєктах. Це дозволяє створювати та поширювати власні модулі та бібліотеки. Розробник може визначити сценарії (scripts) у файлі package.json, що дозволяє виконувати різні команди та дії, такі як запуск тестів, збірка проєкту чи запуск сервера розробки. npm використовує систему версіонування SemVer (Semantic Versioning) для визначення та управління версіями пакетів. Це допомагає забезпечити сумісність та оновлення пакетів у проєкті. Завдяки npm, можна швидко та зручно управляти пакетами та залежностями проєкту, спрощуючи процес розробки та спільної роботи з іншими розробниками.

ElectronJS [4] – це фреймворк для розробки крос-платформових десктопних застосунків з використанням веб-технологій, таких як HTML, CSS і JavaScript. Він дозволяє розробникам створювати нативні десктопні застосунки для операційних систем, таких як Windows, macOS і Linux, використовуючи відомі веб-технології.

Основні характеристики та особливості ElectronJS включають:

- крос-платформовість: ElectronJS дозволяє розробникам створювати десктопні застосунки, які працюють на різних операційних системах без необхідності переписування коду (це забезпечує ефективну розробку та підтримку застосунок на різних платформах);
- використання веб-технологій: ElectronJS базується на веб-технологіях, таких як HTML, CSS і JavaScript, що дозволяє розробникам використовувати веб-навички та інструменти для побудови десктопних застосунків(це також означає, що існуючий веб-код може бути легко переоснащений у десктопний застосунок);
- розширюваність: ElectronJS дозволяє використовувати Node.js для доступу до системних ресурсів та сторонніх бібліотек (це розширює

можливості фреймворка і дозволяє взаємодіяти з операційною системою, виконувати операції на рівні файлової системи, мережі та багато іншого);

- мультиплатформеність: ElectronJS дозволяє пакетувати застосунки в установочні файли для різних операційних систем, що полегшує розповсюдження та встановлення програм (користувачі можуть легко встановлювати та використовувати застосунки, незалежно від їх операційної системи);
- автоматичне оновлення: ElectronJS має вбудовану функцію автоматичного оновлення, яка дозволяє розробникам випускати оновлення застосунків та доставляти їх користувачам безпосередньо на їх пристрої (це спрощує розповсюдження патчів та нових функцій застосунку).
- активна спільнота та екосистема: ElectronJS має широку та активну спільноту розробників, що призводить до постійного розширення функціональності фреймворка (крім того, існує велика кількість сторонніх бібліотек, плагінів та інструментів, які допомагають розширити можливості ElectronJS).

ElectronJS є вибраним фреймворком для розробки десктопних застосунків великою кількістю відомих компаній та продуктів. Один з найвідоміших прикладів використання ElectronJS – Visual Studio Code, один з найпопулярніших редакторів коду, розроблений компанією Microsoft. Visual Studio Code поєднує в собі потужність інтегрованого середовища розробки з веб-технологіями, завдяки ElectronJS. Slack, популярний комунікаційний інструмент для командної роботи, також побудований з використанням ElectronJS. Він надає користувачам можливість спілкуватися, обмінюватися файлами та спільно працювати, використовуючи веб-технології, але з усіма перевагами десктопного-застосунку. Discord, популярна платформа голосового та текстового спілкування для геймерів, також розроблена з використанням ElectronJS. Вона дозволяє гравцям об'єднуватися в групи, спілкуватися під час гри та обмінюватися

інформацією.

ElectronJS допомагає створити потужний десктопний клієнт з використанням веб-технологій, що покращує взаємодію геймерів. Його потужність полягає в тому, що він дозволяє розробникам створювати десктопні застосунки, які поєднують потужність та можливості операційних систем з веб-технологіями. Завдяки цьому, розробники можуть ефективно використовувати свої вебнавички та інфраструктуру, що забезпечує швидкий розвиток та поширення застосунків. ElectronJS відкриває нові горизонти для створення крос-платформових десктопних застосунків, що задовольняють потреби користувачів.

Git [5] є розподіленою системою контролю версій, яка дозволяє відстежувати зміни в файловій системі проєкту. Він зберігає копії файлів на різних етапах розвитку проєкту, що дозволяє повертатись до попередніх версій, робити злиття змін і співпрацювати з іншими розробниками. Існують також інші сервіси контролю версій, такі як Bitbucket, Mercurial та інші, які надають подібні можливості зберігання і спільної роботи над проєктами з використанням різних систем контролю версій.

Git має кілька важливих переваг:

- історія змін: Git зберігає повну історію змін в проєкті, дозволяючи повертатись до будь-якої попередньої версії та відновлювати дані;
- гілки: Git дозволяє створювати гілки, що дозволяє розробляти різні функціональності паралельно, а потім злити їх у головну гілку;
- спільна робота: завдяки Git можна спільно працювати над проєктом з командою розробників, вносячи зміни і злиття до спільного репозиторію;
- відновлення та резервне копіювання: Git дозволяє відновлювати втрачені або пошкоджені файли з попередніх версій та зберігати резервні копії проєкту;
- легкість інтеграції: Git можна легко інтегрувати з різними інструментами розробки, такими як редактори коду, сервіси збирання та автоматизації, тестувальні системи та інші.

Загалом, Git є потужним інструментом для контролю версій, який дозволяє розробникам ефективно керувати змінами у своїх проєктах, співпрацювати з командою та зберігати повну історію розвитку проєкту. Далі будуть більш подібно розписані найпопулярніші системи контролю версій GitHub та GitLab.

GitHub [] є одним з найпопулярніших хостингових сервісів для зберігання та спільної роботи над Git-репозиторіями. Він надає розробникам цілий набір інструментів і функціональності, які полегшують роботу з Git та сприяють співпраці в команді. Основні особливості та можливості GitHub: репозиторії, відстеження проблем, pull-запити, колаборація та співпраця, веб-інтерфейс та інтеграція з іншими інструментами та спільнота та відкритий код.

GitHub дозволяє створювати публічні та приватні репозиторії для зберігання коду та файлів проєкту. Користувач може легко завантажувати, зберігати та оновлювати власні Git-репозиторії на GitHub.

GitHub має вбудовану систему відстеження проблем (issue tracking), яка дозволяє стежити за помилками, побажаннями та завданнями, пов'язаними з проєктом. Користувач може створювати проблеми, коментувати їх, призначати відповідальних, використовувати мітки та замикати проблеми після вирішення. GitHub надає можливість створювати pull-запити, що дозволяє іншим розробникам вносити зміни до проєкту. Користувач може рецензувати запропоновані зміни, обговорювати їх та злити їх у головну гілку проєкту.

GitHub дозволяє легко співпрацювати з командою розробників. Користувач може запрошувати інших користувачів до своїх репозиторіїв, надавати їм права доступу та використовувати функцію "форк" для створення власних копій проєкту. GitHub надає зручний веб-інтерфейс, який дозволяє переглядати, редагувати та взаємодіяти з репозиторіями. Крім того, він має розширений API та підтримку інтеграції з різними інструментами розробки, такими як CI/CD системи, сервіси автоматичного розгортання та інші.

GitHub є популярною платформою для розробки відкритого програмного забезпечення. Користувач може долучатися до проєктів спільноти, співпрацювати з іншими розробниками та сприяти розвитку відкритого коду.

GitHub (див.рис.5) є потужним інструментом для спільної роботи над проєктами, керування версіями та підтримки відкритого програмного забезпечення. Його зручний інтерфейс, функціональність та підтримка роботи з Git роблять його популярним серед розробників усього світу.

Рисунок 5 – Логотип компанії GitHub

GitLab є іншим популярним хостинговим сервісом для зберігання та спільної роботи над Git-репозиторіями. Він надає широкий спектр функціональності та інструментів для ефективного керування проєктами розробки програмного забезпечення. Основні особливості та можливості GitLab: репозиторії, відстеження проблем, CI/CD, спільна робота, контейнеризація, управління доступом та інтеграція з іншими інструментами.

GitLab дозволяє створювати публічні та приватні репозиторії для зберігання власного коду. Користувач може завантажувати, зберігати та оновлювати свій Git-репозиторії на GitLab. GitLab має вбудовану систему відстеження проблем, де користувач може створювати проблеми, призначати їх відповідальним, коментувати та відстежувати їх стан. Це дозволяє ефективно керувати завданнями та проблемами, пов'язаними з проєктом.

GitLab надає вбудовану підтримку для неперервної інтеграції та постій-

ної доставки (CI/CD). Користувач може налаштувати автоматичну збірку, тестування та розгортання вашого коду з використанням інструментів, таких як GitLab Runner. GitLab дозволяє спільно працювати над проєктом з командою розробників, вносячи зміни та злиття до спільного репозиторію. Користувач може створювати гілки, виконувати злиття (merge) та рецензувати код, щоб забезпечити високу якість вашого програмного забезпечення. GitLab має вбудовану підтримку контейнеризації з використанням Docker. Користувач може створювати, керувати та запускати контейнери з вашим програмним забезпеченням, що спрощує процес розгортання та тестування.

GitLab дозволяє налаштовувати рівні доступу до власного репозиторію та проєкту для кожного користувача. Користувач може надавати права на читання, запис або адміністрування, забезпечуючи контроль над вашим кодом. GitLab має широку підтримку інтеграції з різними інструментами розробки, такими як IDE, системи слідкування за помилками, системи спостереження, сервіси хостингу документації та багато інших. GitLab є потужним інструментом для керування проєктами розробки програмного забезпечення та спільної роботи над Git-репозиторіями. Він надає широкий спектр функціональності, дозволяє автоматизувати процеси розробки та підтримує ефективне спілкування та спільну роботу команди розробників (рис.6).

Рисунок 6 – Логотип компанії GitLab

Для подальшої роботи та безпосередньо реалізації було обрано GitHub, оскільки GitHub є найбільш популярною та широко використовуваною платформою для розміщення та спільної роботи над проєктами з відкритим кодом.

GitHub відомий своєю великою спільнотою розробників, яка активно працює над проєктами з відкритим кодом. Розміщуючи застосунок на GitHub, ми відкриваю його для співпраці та внесення змін іншими розробниками. Це сприяє залученню талановитих людей, які можуть внести свій вклад у вдосконалення застосунку та зробити його кращим. GitHub має інтуїтивно зрозумілий інтерфейс, що сприяє зручній навігації та спілкуванню в межах проєкту. Він надає такі функції, як система відстеження проблем, злиття коду, система контролю версій, можливість створення гілок та багато іншого. Це спрощує управління та спільну роботу над проєктом. GitHub надає розширені інструменти для співпраці між розробниками. Це включає можливість відкриття запитів на злиття (Pull Requests), коментування коду, відстеження змін, спільну роботу над проблемами та багато іншого. Ці інструменти допомагають покращити комунікацію та залучити інших розробників до процесу розробки. GitHub є високо надійною платформою, яка забезпечує зберігання та захист коду. Він має резервне копіювання даних, систему контролю доступу та інші заходи безпеки. Крім того, GitHub має великі обчислювальні ресурси, що дозволяє розміщувати проєкти будь-якого масштабу GitHub є платформою, яку використовують мільйони розробників по всьому світу. Це означає, що він має велику активну спільноту та велику кількість розробників, які вже знайомі з цим сервісом. Це робить GitHub привабливим вибором для співпраці та пошуку допомоги. GitHub має широкі можливості інтеграції з іншими сервісами розробки, такими як CI/CD системи, сервіси автоматизації, інструменти для тестування, системи відстеження проблем і багато іншого. Це спрощує роботу з проєктом та дозволяє автоматизувати деякі процеси розробки.

В цілому, GitHub забезпечує всі необхідні інструменти та можливості для розробки застосунку та співпраці над ним. Ця платформа надає зручні засоби для спільної роботи, управління проектом, контролю версій, а також створює комфортне середовище для спілкування з розробницькою спільнотою.

2.2 Обґрунтування вибору середовищ розробки для розробки десктопного застосунку

IDE (Integrated Development Environment) – це середовище розробки, яке надає інтегровані інструменти та функціональність для розробки програмного забезпечення. Різноманітні IDE призначені для різних мов програмування та платформ, і вони надають розробникам зручне та продуктивне середовище для написання, тестування та налагодження коду.

Visual Studio є інтегрованим середовищем розробки (IDE) розроблене компанією Microsoft. Воно надає розробникам потужні інструменти та функціональність для розробки програмного забезпечення на різних платформах і мовах програмування. Основні особливості Visual Studio включають: мультиплатформну підтримку, мовну підтримку, інтегровані інструменти, розширення та плагіни та командну роботу та засоби спільної розробки.

Visual Studio [7] підтримує розробку застосунків для різних платформ, включаючи Windows, macOS, Linux, iOS, Android та веб-платформи. Це дозволяє розробникам створювати крос-платформні рішення, що працюють на різних пристроях і операційних системах.

Visual Studio підтримує широкий спектр мов програмування, включаючи C#, Visual Basic, C++, F#, Python, JavaScript, TypeScript, PHP та багато інших. Кожна мова має свої властивості та інструменти, які допомагають розробникам ефективно працювати з кодом.

Visual Studio надає розробникам доступ до різних інструментів, таких як редактор коду з функціями автодоповнення та перевірки синтаксису, візуальні конструктори інтерфейсу користувача, налагоджувач, система керування версіями, інструменти тестування та профілювання коду. Ці інструменти сприяють зручній розробці, налагодженню та тестуванню програмного забезпечення.

Visual Studio підтримує розширення сторонніх розробників, які дозволяють розширити функціональність IDE. Розробники можуть створювати власні

розширення, додавати нові шаблони проєктів, інтегрувати зовнішні інструменти та плагіни для полегшення роботи з певними технологіями.

Visual Studio має інтегровані засоби для спільної роботи над проєктами, такі як система контролю версій Team Foundation Server (TFS) або Git, інтеграція з системами для збірки та постачання програмного забезпечення (CI/CD), а також засоби для організації завдань та комунікації в команді розробників.

Visual Studio (рис.7) є одним з найпопулярніших та потужних IDE, яке використовується для розробки різних типів програмного забезпечення, включаючи веб-застосунки, мобільні застосунки, хмарні рішення, штучний інтелект та багато інших. Його широкий функціонал та надійність роботи роблять його популярним серед професійних розробників по всьому світу.

Рисунок 7 – Логотип Visual Studio

IntelliJ IDEA є інтегрованим середовищем розробки (IDE), розробленим компанією JetBrains. Воно спеціалізується на підтримці різних мов програмування і надає розробникам потужні інструменти для продуктивної роботи над проєктами. Основні особливості IntelliJ IDEA включають: мовну підтримку, інтелектуальні інструменти, візуальні редактори та дизайнери, інструменти тестування та інтеграція з системами контролю версій.

IntelliJ IDEA підтримує широкий спектр мов програмування, включаючи Java, Kotlin, Scala, Groovy, JavaScript, TypeScript, SQL, HTML, CSS та інші.

Воно надає розробникам розумний автодоповнювач, перевірку синтаксису, рефакторинги та інші інструменти, які полегшують написання коду.

IntelliJ IDEA має ряд інтелектуальних інструментів, що допомагають розробникам писати якісний код. Вони включають аналіз коду, виявлення помилок, оптимізацію імпорту, автоматичне створення коду, вбудований дебагер та інші функції, які сприяють ефективному програмуванню.

IntelliJ IDEA надає візуальні редактори для різних технологій, таких як Swing, JavaFX, Android XML, HTML та CSS. Ці редактори дозволяють розробникам швидко створювати та налаштовувати інтерфейс користувача своїх застосунків.

IntelliJ IDEA має вбудовану підтримку для написання і запуску автоматизованих тестів. Вона підтримує різні фреймворки тестування, такі як JUnit, TestNG, Spock, і забезпечує можливості для легкого створення, запуску та аналізу тестових сценаріїв.

IntelliJ IDEA інтегрується з різними системами контролю версій, такими як Git, Subversion, Mercurial, CVS, і дозволяє розробникам легко працювати зі своїми репозиторіями, виконувати коміти, переглядати історію змін та розв'язувати конфлікти.

IntelliJ IDEA (рис.8) широко використовується професійними розробниками завдяки своїй потужності, продуктивності та широкому спектру функціональних можливостей. Воно надає зручне середовище розробки для різних проєктів, від невеликих програм до складних корпоративних застосунків.

Рисунок 8 – Логотип компанії IntelliJ IDEA.

PyCharm є інтегрованим середовищем розробки (IDE), розробленим компанією JetBrains, спеціалізованим на розробці програм на мові програмування Python. Він надає розробникам потужні інструменти та функції, які сприяють продуктивності та ефективності в розробці Python-проєктів. Основні особливості PyCharm включають: підтримка python, розширені інструменти розробки, візуальні редактори та дизайнери, інтеграція з іншими інструментами та підтримка розробки веб-застосунків.

PyCharm надає повну підтримку Python, включаючи версії 2.x і 3.x. Він має розумний автодоповнювач, перевірку синтаксису, рефакторинги, візуальний дебагер та багато інших інструментів, що полегшують написання коду на Python.

PyCharm надає широкий спектр інструментів для полегшення розробки, таких як автоматичне створення коду, виявлення помилок, аналіз коду, оптимізація імпорту, вбудований дебагер, система управління пакетами та інші.

PyCharm має вбудовані редактори для різних типів файлів, включаючи Python-файли, HTML, CSS, JavaScript, SQL та інші. Ці редактори надають можливість редагувати код з підсвічуванням синтаксису, автоматичним вирівнюванням, перевіркою правопису та іншими функціями.

PyCharm підтримує інтеграцію з різними засобами розробки, такими як системи контролю версій Git, Mercurial, Subversion, системи відстеження помилок, засоби віртуалізації, засоби для розробки веб-застосунків та багато інших.

PyCharm має спеціальні інструменти для розробки веб-застосунків на основі фреймворків, таких як Django, Flask, Pyramid, та інших. Він надає можливість автоматичного створення проєктів, візуальне моделювання баз даних, налагоджування серверної частини та інші корисні функції.

PyCharm є потужним інструментом для розробки на мові програмування Python, який допомагає розробникам створювати якісний і ефективний код, прискорюючи процес розробки та полегшуючи роботу з проєктами (рис.9).

Рисунок 9 – Логотип компанії PyCharm

XCode є інтегрованим середовищем розробки (IDE), розробленим компанією Apple, і використовується для розробки програм для платформ iOS, macOS, watchOS та tvOS. Це основний набір інструментів для розробки програмного забезпечення для усіх пристроїв Apple.

Основні особливості XCode включають: розробку під платформи Apple, інтерфейс для розробки, інструменти для тестування, інтеграцію з іншими інструментами та симулятори пристроїв.

XCode надає розробникам засоби для створення програм для пристроїв Apple, таких як iPhone, iPad, Mac, Apple Watch та Apple TV. Воно підтримує мови програмування Swift та Objective-C і надає можливості для створення різноманітних застосунків, від мобільних застосунків до застосунків для робочих станцій.

XCode має інтуїтивний інтерфейс, який спрощує процес розробки програм. Воно надає редактор коду з підсвічуванням синтаксису, автодоповнюванням та вбудованим дебагером. Також воно має інструменти для створення і налагодження інтерфейсу користувача, такі як Interface Builder.

XCode надає інструменти для автоматизованого тестування програм. Воно підтримує різні фреймворки тестування, такі як XCTest, що дозволяють розробникам виконувати тестування одиниць, функціональне тестування та тестування інтерфейсу користувача.

XCode інтегрується з іншими інструментами розробки, такими як система контролю версій Git, і надає можливість спільної роботи в команді. Воно також має інструменти для аналізу коду, профілювання застосунків та створення документації.

XCode має вбудовані симулятори пристроїв, що дозволяють розробникам тестувати свої програми на різних пристроях та платформах, необхідних для їх розробки. Це дозволяє виробникам програм забезпечити широку сумісність і перевірити роботу програми на різних пристроях перед релізом.

XCode (рис.10) є незамінним інструментом для розробки програмного забезпечення для платформ Apple, забезпечуючи розробникам усі необхідні інструменти та ресурси для створення якісних та продуктивних застосунків.

Рисунок 10 – Логотип XCode

Visual Studio Code (VS Code) – це безкоштовне, відкрите та легковагове інтегроване середовище розробки (IDE), розроблене компанією Microsoft. Воно призначене для розробки програмного забезпечення на різних мовах програмування, включаючи JavaScript, TypeScript, Python, Java, C++, і багато інших. Основні особливості Visual Studio Code включають: розширення та налаштування, кодовий редактор з функціональністю IDE, інтеграція з системами

контролю версій, вбудована термінал, зручний пошук та навігація, відлагодження та крос-платформеність. VS Code має широкий вибір розширень, які дозволяють розширити його функціональність для відповідності потребам. Розширення включають синтаксичне підсвічування, автодоповнення, налаштування компіляції та інші інструменти, що полегшують розробку. VS Code надає потужний кодовий редактор з функціями, такими як підсвічування синтаксису, автодоповнення, перетягування та копіювання рядків коду, а також інструменти для рефакторингу та налагодження. VS Code має підтримку для популярних систем контролю версій, таких як Git. Користувач може виконувати операції комітів, відгалужування, злиття та інші дії безпосередньо з IDE. VS Code має вбудований термінал, що дозволяє виконувати команди відразу з редактора. Це зручно для виконання різних команд, запуску програм та взаємодії з командним рядком без виходу з середовища розробки.

VS Code надає потужні інструменти для пошуку та навігації по коду. Користувач може швидко знаходити файли, функції, змінювати місця розташування та використовувати закладки для швидкого доступу до важливих частин коду. VS Code має вбудовані інструменти для відлагодження, що дозволяють крокувати по коду, перевіряти значення змінних та аналізувати стек викликів. Це полегшує виявлення та виправлення помилок у програмі. VS Code підтримує Windows, macOS та Linux, що дозволяє розробникам працювати на будь-якій платформі, яка їм зручна. Visual Studio Code є дуже популярним середовищем розробки завдяки своїй легкості, розширюваності та потужним інструментам, що полегшують розробку програмного забезпечення на різних мовах програмування. Він став вибором багатьох розробників у всьому світі.

Для реалізації програмної частини кваліфікаційної роботи бакалавра було обрано як IDE – Visual Studio Code (VS Code). VS Code має широкий вибір розширень, які дозволяють розширити його функціональність. Це означає, що можна знайти розширення, які підходять для потреб розробки, наприклад, розширення для підтримки React та ElectronJS. Завдяки цим розширенням можна підтримувати і покращувати код, використовуючи найновіші інструменти

та технології. VS Code має простий та інтуїтивно зрозумілий інтерфейс, який полегшує розробку. Його швидка реакція на введення та вбудовані функції, такі як автодоповнення, допомагають прискорити процес написання коду. Це робить його дуже зручним для щоденної роботи. VS Code підтримує Windows, macOS та Linux, що означає, що можна працювати на будь-якій зручній платформі. Це дає гнучкість та можливість розробляти застосунок на будь-якому комп'ютері, незалежно від операційної системи. VS Code має вбудовану підтримку для систем контролю версій, зокрема Git. Це дозволяє легко працювати з репозиторіями Git, виконувати коміти, переглядати різницю в коді та керувати гілками, не виходячи з редактора. Що в свою чергу значно полегшує управління та спільну роботу з кодом. VS Code має велику та активну спільноту розробників, що дозволяє швидко знайти відповіді на свої питання, знайти корисні ресурси та отримати підтримку в разі необхідності. Також, завдяки активній спільноті, створення та публікація розширень стає простішим та ефективнішим процесом.

Загалом, VS Code має потужний набір функцій, простий у використанні та інтерфейс який добре налаштовується, а також розширюваність, що дозволяє ефективно (див.рис.11).

Рисунок 11 – Логотип Visual Studio Code

3. РЕАЛІЗАЦІЯ ДЕСКТОПНОГО ЗАСТОСУНКУ ДЛЯ ОРГАНІЗАЦІЇ РОБОЧОГО ПРОСТОРУ СТУДЕНТА

3.1 Опис етапів розробки та реалізації десктопного застосунку

Для виконання кваліфікаційної роботи бакалавра (КРБ) на тему «Розробка десктопного застосунку щодо організації робочого простору студента» було проаналізовано велику кількість технології та середовища розробки. Безпосередньо для розробки було обрано JavaScript [8] як основну мову програмування, для кросплатформованості було обрано ElectronJS, а для зручної роботи з DOM та для написання коду– бібліотеку React.

Обираючи Visual Studio Code (VS Code) як середовище для подальшої розробки застосунку, було враховано декілька факторів – VS Code є одним з найпопулярніших редакторів серед розробників, що означає наявність активної та розширеної спільноти. Це означає, що є багато ресурсів, документації, розширень та підтримки, які допоможуть у розробці застосунку. VS Code підтримує всі основні операційні системи, такі як Windows, macOS і Linux. Це дозволить розробляти застосунок на будь-якій платформі за допомогою одного і того ж редактора. VS Code має потужну систему розширень, що дозволяє додати функціональність, яка потрібна для свого застосунку. Існує безліч розширень, що полегшують роботу з різними мовами програмування, фреймворками, інструментами та іншими технологіями. VS Code має потужні інструменти для роботи з системою контролю версій Git. Це дозволяє в свою чергу зручно відстежувати та керувати змінами у застосунку, використовуючи всі можливості Git.

VS Code має інтуїтивний і зрозумілий інтерфейс, що робить його легким у використанні навіть для початківців. Він надає швидкий доступ до основних функцій редактора та надає зручний розклад панелей та вікон. VS Code надає можливості відладки для різних мов програмування та платформ. Це дозволяє зручно виявляти та виправляти помилки у застосунку, полегшуючи процес розробки. VS Code має велику кількість розширень, що покривають різні аспекти

розробки, включаючи розширену підтримку React, TypeScript, Git, форматування коду та багато іншого. Це дозволяє налаштувати редактор під будь-які потреби та працювати з комфортом.

3.2 Використання React для створення інтерфейсу та архітектури десктопного застосунку

Процес встановлення Node.js може залежати від операційної системи, але основні кроки залишаються однаковими. Також для роботи необхідно встановити React, а потім створити застосунок. По-перше необхідно за допомогою терміналу або командного рядка перейти до папки в якому буде розміщений проєкт React. Далі, щоб створити безпосередньо проєкт з використанням шаблону Create React App варто прописати команду:

```
npx create-react-app my-app
```

Дана команда встановить необхідні залежності та створить початкову структуру проєкту. Далі необхідно перейти до папки проєкту за допомогою команди:

```
cd electron-react-diary
```

Для запуску застосунку на основі React можна використати наступну команду:

```
npm start
```

Процес додавання ElectronJS до React-застосунку можна розділити на кілька основних кроків. Спочатку потрібно створити React-застосунок за допомогою інструментів, таких як Create React App або власного шаблону. Цей

крок включає створення базової структури проєкту та налаштування залежностей. Наступним кроком є встановлення ElectronJS у проєкт. Це можна зробити за допомогою менеджера пакетів npm або yarn, виконавши відповідну команду для встановлення ElectronJS. Після встановлення ElectronJS потрібно налаштувати конфігураційні файли. Зазвичай це включає створення файлу main.js, який є точкою входу для ElectronJS-застосунку. В цьому файлі можна визначити вікно застосунку, налаштувати ресурси та обробку подій. Для інтеграції ElectronJS з React необхідно змінити файл index.js або аналогічний файл, який є точкою входу для React-застосунку. Візуалізація вікна Electron реалізується за допомогою функції, яка задає розмір самого вікна Electron та завантажує вміст цього вікна з файлу index.html.

```
const { app, BrowserWindow } = require('electron');

function createWindow() {
 const mainWindow = new BrowserWindow({
  width: 800,
  height: 600,
  webPreferences: {
   nodeIntegration: true
  }
 });

 mainWindow.loadFile('index.html');
}

app.whenReady().then(createWindow);
```

Також можна використовувати electron-renderer або electron-preload для забезпечення зв'язку між ElectronJS та React. За допомогою скриптів збирання, таких як electron-builder або electron-packager, можна зібрати ElectronJS застосунок для різних платформ (Windows, macOS, Linux). Після збирання можна запустити застосунок на обраній платформі та перевірити його працездатність.

Вище описана загальна схема процесу додавання ElectronJS до React-за-

стосунку. Важливо враховувати, що під час реалізації можуть виникати особливості або додаткові кроки залежно від конкретних вимог та налаштувань проєкту.

Управління залежностями та налаштування запуску було здійснено через package.json. У цьому файлі були вказані всі необхідні залежності, включаючи React та ElectronJS, а також скрипти для запуску, збірки та розгортання проєкту. Завдяки поєднанню цих залежностей та налаштуванням у package.json, було досягнуто сумісності між React та ElectronJS.

```
"scripts": {
"start": "cross-env BROWSER=none react-scripts start",
"dev": "concurrently -k \"cross-env BROWSER=none npm start\" 
\"npm:electron .\"",
"electron": "electron .",
},
```

React використовується для розробки користувацького інтерфейсу, створення компонентів та управління станом застосунку, тоді як ElectronJS забезпечує обгортку довкола React-застосунку, дозволяючи йому працювати як десктопний застосунок. Це архітектурне рішення дозволяє ефективно поєднати можливості React для розробки користувацького інтерфейсу з перевагами ElectronJS для створення кросплатформових десктопних застосунків.

3.3 Проєктування та організація загальної архітектури десктопного застосунку

Архітектура застосунку побудована таким чином, що компоненти мають бути реюзабельними та зрозумілими для читання. Це досягається за допомогою належної структури та назв компонентів. При проєктуванні архітектури, особлива увага приділяється назвам та місцезнаходженню компонентів, що допомагає зробити розуміння та підтримку застосунку більш простим для майбутніх розробників та користувачів.

Використання зрозумілих та описових назв для компонентів важливо, оскільки це дозволяє легко ідентифікувати їх функціональність та призначення. Наприклад, якщо компонент називається "Header", це надає вказівку, що він відповідає за відображення заголовка на сторінці. Зрозумілі назви спрощують спілкування між розробниками та полегшують розуміння структури застосунку.

Крім того, місцезнаходження компонентів також має значення. Хороша практика полягає в групуванні компонентів за їх функціональністю або логічною структурою. Наприклад, всі компоненти, пов'язані з навігацією, можуть бути розташовані в одній папці або модулі. Це спрощує пошук та зміну компонентів, а також полегшує орієнтацію для нових розробників, які приєднаються до проєкту. Розташування компонентів та зрозумілі назви допомагають зробити код більш читабельним та організованим. Це сприяє легкості утримання та розширення застосунку, а також підвищує швидкість розробки, оскільки новим розробникам легше зорієнтуватися та внести необхідні зміни. Загалом, уважне ставлення до назв та місцезнаходження компонентів є ключовим фактором для забезпечення чистої та організованої архітектури, яка сприяє підтримуваності та розширюваності застосунку. орієнтиром при написанні архітектури застосунку були найсучасніші практики побудови проєкту, а також книга «Чистий Код» Роберта Мартіна.

Роберт Мартін [9] пропонує ряд порад та практик, які можна застосовувати для поліпшення якості коду. Основні поради, які можуть бути використані для побудови чистого коду у даному застосунку, це:

- правильне форматування, використання зрозумілих імен змінних та функцій, адекватний розмір функцій та класів;
- розбиття функцій на менші компоненти з однією конкретною відповідальністю для підвищення зрозумілості та легкості підтримки;
- уникання зайвих коментарів, пріоритет на зрозумілий код, що сам по собі пояснює свої інтенції;
- високий рівень тестового покриття, включаючи одиниці тестування,

інтеграційні тести та автоматизовані тести;

– уникання дублювання коду шляхом використання функцій, класів або шаблонів;
– використання систем контролю версій, таких як Git, для ефективного керування та відстежування змін коду;
– розумне планування та організація коду, використання шаблонів проєктування та принципів SOLID для створення гнучкої та легко розширюваної архітектури.

Читабельніший код зменшує шанси на помилки, спрощує співпрацю з іншими розробниками та полегшує підтримку проєкту. Також, використання тестування та систем контролю версій дозволяє ефективно працювати над розробкою та забезпечувати стабільність кодової бази (рис.12).

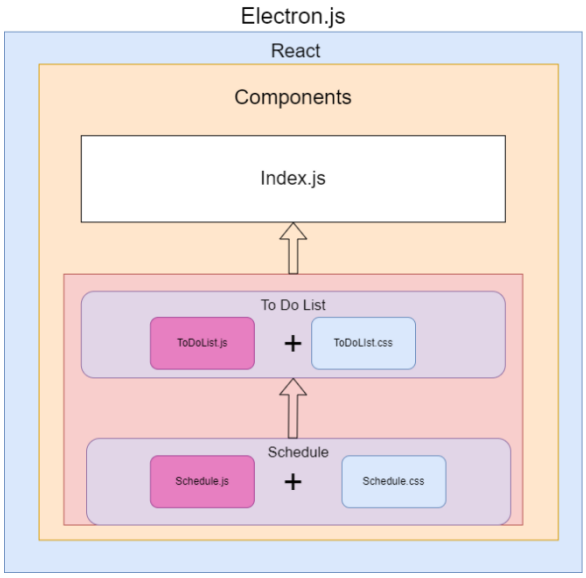

Рисунок 12 – Принцип компіляції застосунку

Кореневий елемент index.js є основним модулем використовуваного React фреймворку, він відповідає за ініціалізацію та маніпуляції з віртуальним DOM (Document Object Model) деревом. Загальним призначенням кореневого елемента є початкова точка входу для React-застосунку, де віртуальний DOM буде рендеритися та оновлюватися (рис.13).

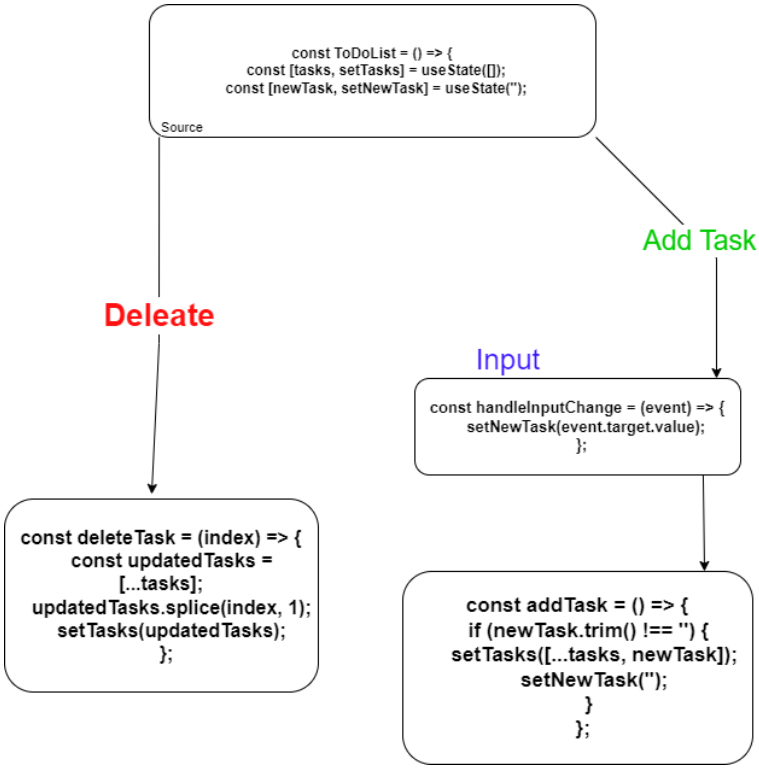

Рисунок 13 – Схематичне зображення роботи компонента ToDoList

У контексті розробки з React, index.js містить наступний код:

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import reportWebVitals from './reportWebVitals';
import ToDoList from './components/ToDoList';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <ToDoList />
  </React.StrictMode>
);
```

У цьому коді використовуються імпорти React та ReactDOM для забезпечення доступу до необхідних функцій та класів. Компонент App, який є головним компонентом застосунку, передається в функцію ReactDOM.render(),

яка відповідає за рендеринг компонента в кореневий елемент з ідентифікатором 'root'. Такий підхід дозволяє React взаємодіяти з реальним DOM, шляхом віртуального DOM, що приводить до зниження затрат на обновлення сторінки та покращує продуктивність застосунку. Отже, кореневий елемент index.js є початковою точкою входу React-застосунку, де відбувається ініціалізація та рендеринг віртуального DOM дерева, забезпечуючи ефективну маніпуляцію з компонентами та оновлення відповідного вмісту на сторінці

З метою здійснення систематизації та виконання різноманітних завдань, розроблений був компонент ToDoList. Цей компонент є структурною одиницею, яка призначена для зберігання та управління списком завдань. Його архітектура сприймає наступну форму:

```
const ToDoList = () => { const [tasks, setTasks] = useState([]);
const [newTask, setNewTask] = useState('');
```

Функціональний компонент ToDoList визначається за допомогою стрілочної функції. Використовується хук useState для створення двох змінних стану: tasks та newTask. Змінна tasks зберігає масив завдань, а newTask зберігає поточне значення поля вводу для додавання нових завдань.

```
 const handleInputChange = (event) => {
setNewTask(event.target.value); };
```

Функція handleInputChange приймає подію event і встановлює нове значення для змінної стану newTask на основі значення поля вводу. Це відбувається за допомогою setNewTask(event.target.value), де event.target.value представляє поточне значення поля вводу.

```
const addTask = () => {
 if (newTask.trim() !== '')
 { setTasks([...tasks, newTask]);
 setNewTask(''); } };
```

Функція addTask викликається, коли користувач натискає кнопку для додавання завдання. Спочатку перевіряється, чи не є поточне значення newTask пустим або містить тільки пробіли (newTask.trim() !== ''). Якщо умова виконується, тоді завдання додається до масиву tasks за допомогою оператора розширення ([...tasks, newTask]), і змінна newTask очищається (setNewTask('')). (рис.14)

Рисунок 14 – UI компонента ToDoList

```
const deleteTask = (index) => {
 const updatedTasks = [...tasks];
 updatedTasks.splice(index, 1);
 setTasks(updatedTasks); };
```

Функція deleteTask реалізація якої в застосунку представлена на рис.15 видаляє завдання зі списку за певним індексом. Спочатку створюється копія масиву tasks за допомогою оператора розширення ([...tasks]). Далі використовується метод splice для видалення одного елемента з масиву за вказаним індексом (updatedTasks.splice(index, 1)). Нарешті, оновлений масив updatedTasks встановлюється як новий стан за допомогою setTasks(updatedTasks).

Рисунок 15 – Візуалізація доданих задач з функціоналом їх видалення

Таким чином, компонент ToDoList має можливість додавати нові завдання до списку та видаляти існуючі завдання. Зміни в стані tasks та newTask

автоматично відображаються на веб-сторінці завдяки принципу перерендерингу компонентів у React при зміні стану. Реалізація розкладу занять відображена у компоненті Schedule.js, (рис.16) що представляє собою структуровану схему розподілу академічних занять, відображену у віртуальному дереві об'єктів уявного моделювання (Virtual DOM) компонента. Компонент включає розташовану в ньому кнопку "Connect", яка надає можливість зв'язатися зі заняттями через платформу Zoom (рис.17).

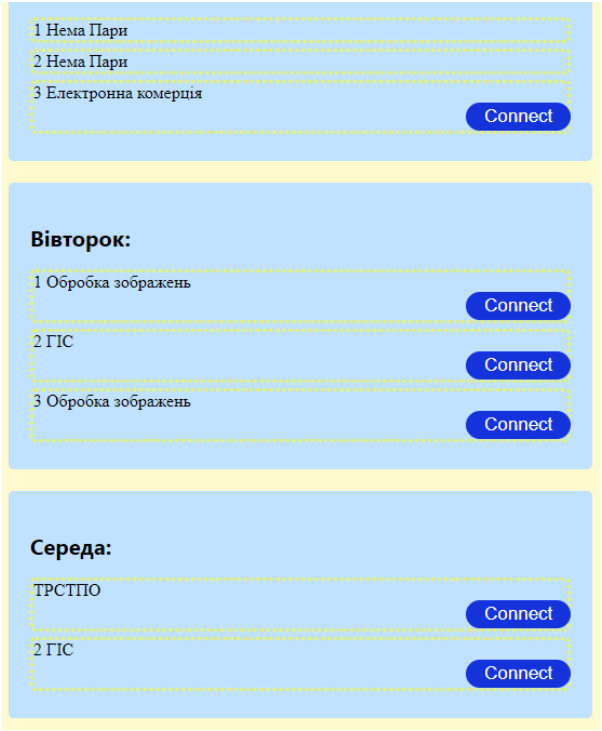

Рисунок 16 –Відображення графіку занять

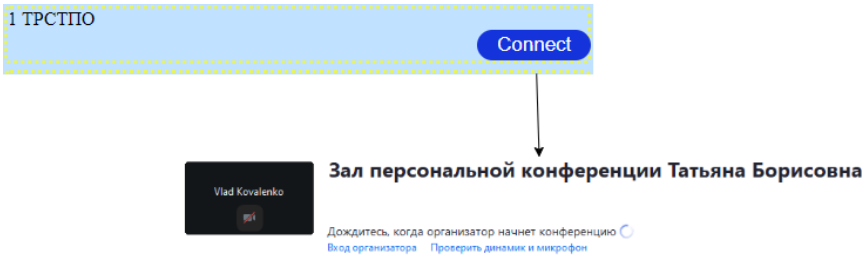

Рисунок 17 – Результат натискання на кнопку «Connect»

У процесі розробки було створено прототип бази даних, яка включає такі таблиці, як "Courses", "Students", "Schedule" і "Enrollments". Цей прототип бази даних дозволяє зберігати та організовувати дані, необхідні для ефективного функціонування застосунки розкладу.

Впровадження бази даних в застосунок розкладу принесе кілька переваг. По-перше, база даних дозволить ефективно зберігати, організовувати та керувати інформацією про курси, студентів, розклад занять та записи про зареєстрованих студентів на курси. Завдяки базі даних, реалізується можливість, редагувати та видаляти дані безпосередньо з застосунку, що спростить процес управління розкладом.

По-друге, база даних забезпечить швидкий доступ до потрібних даних, оскільки можна виконувати оптимізовані SQL-запити, що прискорить обробку даних та покращить продуктивність застосунку. Запити до бази даних дозволять здійснювати фільтрацію, сортування та пошук інформації з розкладу, що полегшить навігацію та взаємодію користувача з застосунком.

Крім того, база даних дозволить забезпечити стійкість та цілісність даних. Ви зможете встановити правила цілісності, які гарантують правильність даних в базі, а також забезпечити можливість резервного копіювання та відновлення даних для забезпечення безпеки і надійності.

Застосування бази даних до застосунку розкладу покращить його функціональність, ефективність та зручність використання. Це дозволить користувачам швидко та легко отримувати доступ до актуальної інформації про курси та розклад занять, а розробникам – здійснювати ефективне керування даними та розширення функціональності застосунку. На рис.18 наведено макет бази даних.

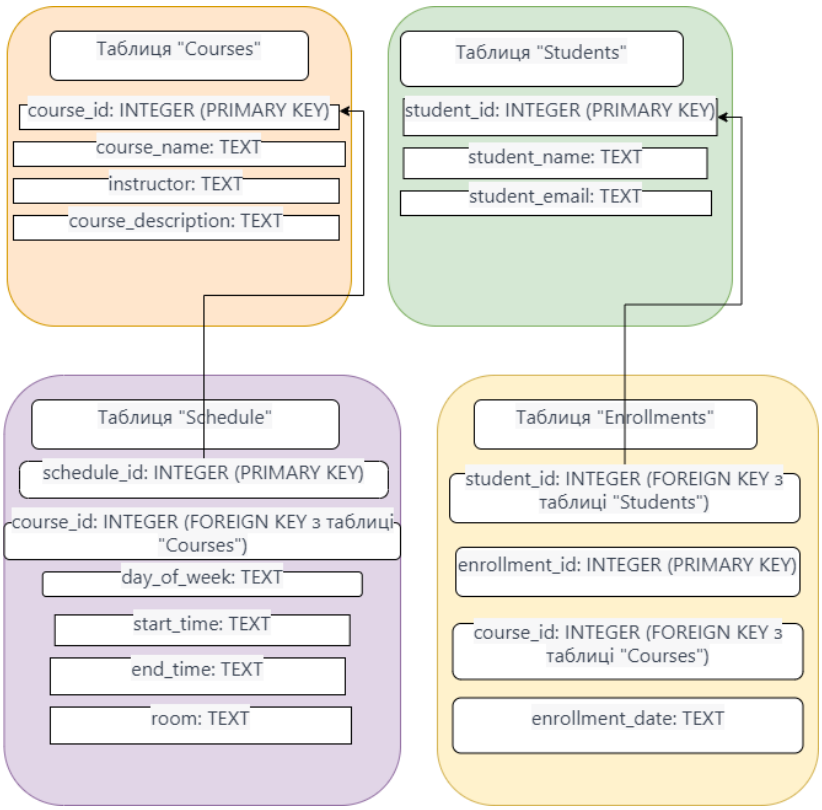

Рисунок 18 – Макет структури бази даних

Ця структура передбачає зв'язки між таблицями. Наприклад, поле "course_id" в таблиці "Schedule" є зовнішнім ключем (FOREIGN KEY), який посилається на поле "course_id" в таблиці "Courses". Таким чином, реалізується можливість зв'язати розклад з конкретним курсом.

Аналогічно, поле "student_id" в таблиці "Enrollments" є зовнішнім ключем, який посилається на поле "student_id" в таблиці "Students", а поле "course_id" також є зовнішнім ключем, який посилається на поле "course_id" в таблиці "Courses". Таким чином, ви можете відстежувати записи про зареєстрованих студентів на конкретні курси.

Застосунок на етапі розробки проходить кілька кроків до свого повного зібрання і запуску. Цей процес починається з використання команди "npm start". Дана команда запускає локальний сервер React, який слухає на порту

3000. Локальний сервер дозволяє переглядати та тестувати застосунок в реальному часі під час розробки.

Для того, щоб отримати десктопний застосунок, необхідно використати команду "npm electron". Ця команда відкриває вікно Electron, яке виконує React-застосунок в середовищі десктопної платформи. Воно поєднує можливості веб-технологій React з потужністю та можливостями операційної системи, що дозволяє-створити крос-платформений десктопний застосунок. Цей процес дозволяє переконатися, що застосунок працює належним чином як у веб-браузері, так і в десктопному середовищі. Розробник може перевірити його функціональність, взаємодію з користувачем та забезпечити, що все працює правильно перед його повним зібранням та випуском.

Застосунок можна умовно поділити на два основні компоненти: ToDoList та Schedule. ToDoList відповідає за управління списком задач. Цей компонент дозволяє користувачам додавати нові завдання, видаляти їх, позначати як виконані та вносити зміни до існуючих задач. Він забезпечує організацію та систематизацію завдань, що допомагає студентам планувати свої дії та досягати поставлених цілей. Другим компонентом є Schedule, – який відповідає за розклад пар та підключення до них. Цей компонент дозволяє студентам переглядати розклад своїх занять, отримувати інформацію про час, місце та викладачів. Крім того, він надає можливість швидкого підключення до відповідних пар за допомогою вбудованого планувальника. Це дозволяє студентам ефективно організовувати свій робочий час та не пропускати важливі заняття.

Поділ застосунка на ці два компоненти, допомагає користувачам зосередитися на конкретних аспектах їх робочого простору. Вони можуть фокусуватися на управлінні задачами та виконанні планів, одночасно не втрачаючи орієнтацію у розкладі занять. Це забезпечує зручну та ефективну організацію робочого простору студента, підвищує продуктивність та допомагає досягати намічених цілей (див.рис.19).

Рисунок 19 – Інтерфейс застосунку для організації робочого простору студента

Одним із вибраних рішень для написання компонентів проєкту було виділення CSS в окремі файли від JS файлів. Цей підхід, відомий як "розділення зв'язків" (separation of concerns). По-перше, розміщення CSS в окремих файлах дозволяє чітко визначити роль і функціональність кожного файлу. JS файл відповідає за логіку та поведінку компоненту, тоді як CSS файл відповідає за його зовнішній вигляд. Це забезпечує збереження принципу однієї відповідальності та полегшує розуміння та підтримку коду. По-друге, розділення CSS в окремі файли дозволяє легко перевикористовувати стилі між різними компонентами або навіть між різними проєктами. Це сприяє підтримці єдиного стилю та забезпечує швидку та просту модифікацію стилів. По-третє, відокремлення CSS від JS дозволяє зосередитися на кожному типі коду окремо, покращуючи читабельність та обслуговування. Розподіл функціональності на два окремих файли спрощує зрозуміння та зміну коду в майбутньому.

Отже, для забезпечення чистоти коду, простоти розуміння, перевикористання стилів та уникнення недоліків, пов'язаних з використанням готових CSS бібліотек, було вирішено розділити CSS в окремі файли від JS файлів у застосунку. Процес створення Git репозиторія та його додавання на GitHub можна розбити на кілька кроків. По-перше, необхідно увійти до свого облікового запису на GitHub. На верхній панелі сторінки необхідно натиснути кнопку "+", а потім у випадаючому меню обрати "New repository"(рис. 20).

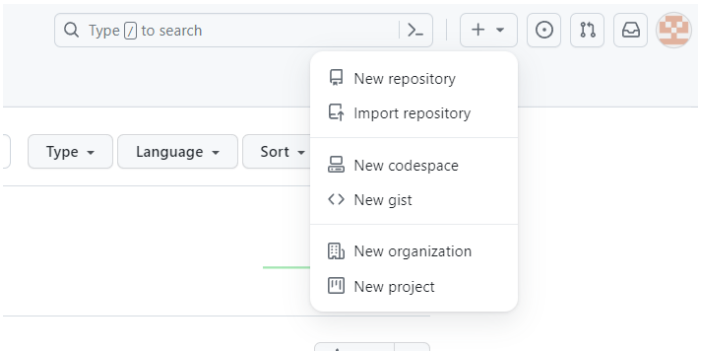

Рисунок 20 – Вкладка для створення нового репозиторію

Далі варто ввести назву для репозиторію. Назва повинна бути унікальною на GitHub. За необхідності додати опис до репозиторію.

Наступним кроком є вибір виду видимості репозиторію: Public (публічний) або Private (приватний). Публічний репозиторій буде доступний для перегляду та внесення змін усім користувачам GitHub, приватний репозиторій – буде доступний лише для обраних користувачів. Опція "Initialize this repository with a README" (додається за бажанням). Дана опція дозволяє створити файл README.md, який можна використовувати для опису проєкту.

Також опціонально можна обрати тип лицензії (ліцензія встановлює правила використання коду розробника другими користувачами), опцію "Add .Gitignore" (це дозволить вибрати попередньо налаштований файл .Gitignore, який виключить певні файли та папки з контролю версій Git), опцію "Add a license" (дозволить вибрати попередньо налаштовану ліцензію для вашого

проєкту). Після заповнення всіх необхідних полів необхідно натиснути кнопку "Create repository" (рис.21).

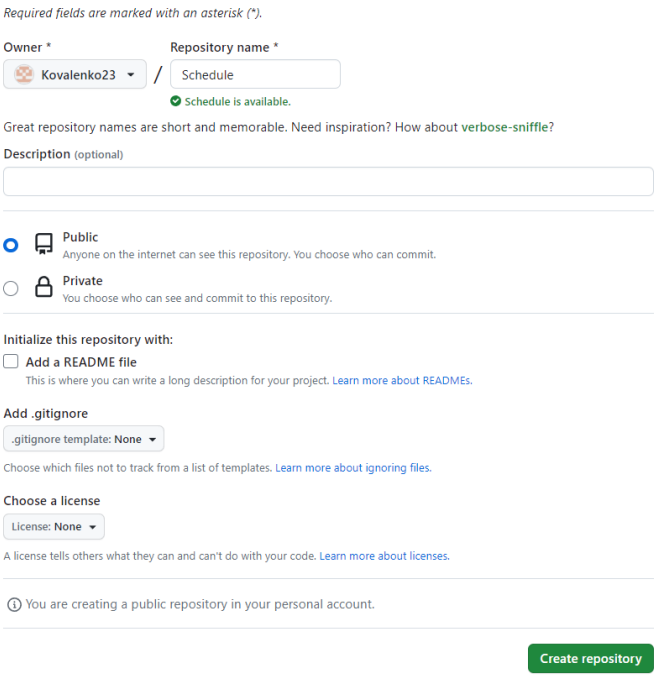

Рисунок 21 – Меню налаштування нового репозиторію

Створений репозиторій на GitHub представлено нижче (рис. 22)

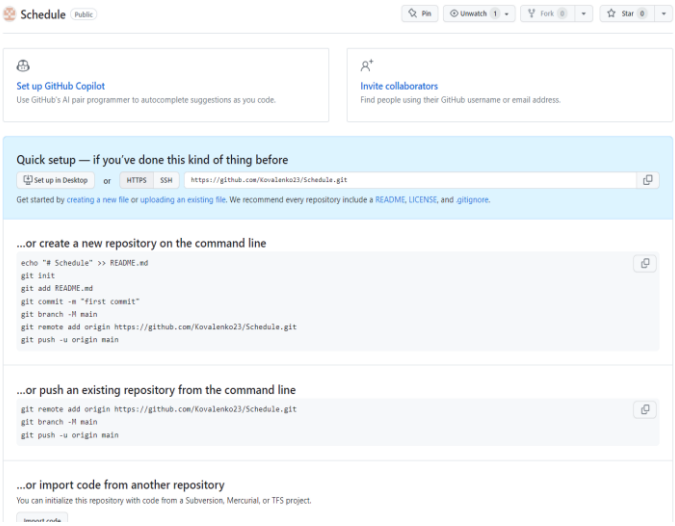

Рисунок 22 – Створений репозиторій

Слід зауважити, що створений репозиторій на GitHub буде пустим. Додавати файли, комітити зміни та працювати зі створеним репозиторієм в подальшому можна буде використовуючи команди Git.

Щоб ініціалізувати Git репозиторія, необхідно відкрити термінал або командний рядок в потрібній папці, де можна створити репозиторій. Команда Git init,-створить порожній локальний репозиторій.

Щоб додавати файли до репозиторію, слід скопіювати або створити файли, які необхідно додати до репозиторію, у відповідну папку. Щоб додати всі файли з поточної папки до стадії, варто виконати команду Git add.

Команда Git commit -m "Перше комітування", допоможе зберегти зміни у локальному репозиторії зі зрозумілим повідомленням про коміт.

Щоб додавати віддалений репозиторій, необхідно після створення репозиторію на GitHub, скопіювати URL-адресу репозиторію. Потім, повернутися до терміналу або командного рядка і виконати команду Git remote add origin <URL>, де <URL> – це URL-адреса віддаленого репозиторію на GitHub.

Команда Git push -u origin master, дозволяє відправити локальні зміни на віддалений репозиторій на GitHub. Після виконання цих кроків, Git репозиторій буде успішно створений та доданий на GitHub. Розробник може бачити свої зміни та працювати з репозиторієм як на локальному комп'ютері, так і на веб-інтерфейсі GitHub. (рис.23).

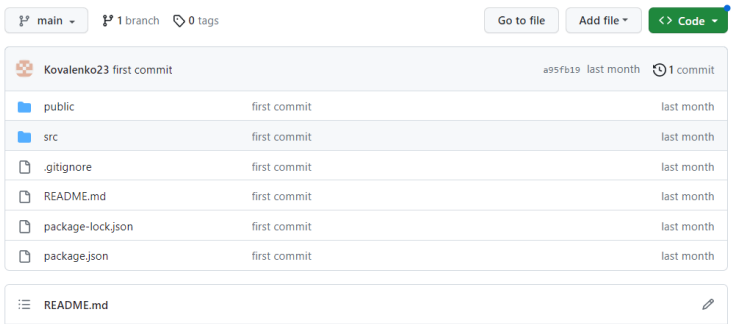

Рисунок 23 – Репозиторій з завантаженим застосунком

3.4 Опис інсталяційного тестування застосунку

Інсталяційне тестування застосунку складається з наступних етапів:

- успішність запуску програми після встановлення;
- розташування програми в файловій системі за замовчуванням;
- розташування програми в файловій системі, якщо шлях збереження змінений користувачем;
- наявність ярликів на робочому столі, узагалі можливість туди його імпортувати;
- чи є встановлений компонент в меню Пуск → Програми;
- встановлення програми для поточного користувача / для всіх користувачів комп’ютера.

ВИСНОВОК

У рамках даної кваліфікаційної роботи бакалавра був розроблений десктопний застосунок з використанням ElectronJS та React для організації робочого простору студента. Застосунок має функціонал, що дозволяє переглядати розклад пар та швидко підключатися до них, а також використовувати To Do List для керування задачами.

Проведений аналіз існуючих аналогів на ринку, таких як Google Calendar, Microsoft Outlook, Todoist та Notion, показав, що застосунок має свої переваги. Він поєднує зручний інтерфейс, розширені можливості та відкритий код, що дозволяє іншим студентам доповнювати та вносити зміни до застосунку через платформу GitHub.

Під час розробки було використано React, що забезпечує швидку та ефективну розробку інтерфейсу користувача. ElectronJS дозволив створити кросплатформовий десктопний застосунок, який працює на різних операційних системах, забезпечуючи високий рівень доступності.

Використання Git та платформи GitHub дало можливість ефективно керувати версіями коду, спільно працювати над проєктом та залучати спільноту до розвитку застосунок. Це сприяє швидкому виявленню та виправленню помилок, а також розширенню функціоналу за допомогою внесення змін від інших розробників.

В результаті розробки було створено десктопний застосунок, який допомагає студентам організувати робочий простір, планувати завдання та керувати розкладом занять. Він поєднує зручність використання, розширені можливості та відкритий код, що робить його привабливим вибором для студентів. В рамках розробки десктопного застосунку для організації робочого простору студента було використано сучасні технології та інструменти, такі як ElectronJS і React. Це дозволило створити потужний та функціональний застосунок, який забезпечує зручний доступ до розкладу пар, швидке підключення до них та ефективне керування задачами.

Однією з ключових переваг розробленого застосунку є його відкритий та зрозумілий код, що сприяє співпраці та внесенню змін в проєкт з боку інших студентів через платформу GitHub. Це забезпечує широкі можливості розвитку застосунку та сприяє створенню активної спільноти розробників.

Важливим аспектом під час розробки було дотримання принципів чистого коду, які були взяті з книги "Чистий код" Роберта Мартіна. Ці поради та практики допомогли створити добре структурований, зрозумілий та підтримуваний код, що сприяє швидкому розробленню нових функцій та виявленню та виправленню помилок.

Застосунок показав свою ефективність та користь для студентів, надаючи їм зручні та потужні інструменти для організації їхнього робочого простору. Широкий функціонал, простий та інтуїтивний інтерфейс, а також можливість розширення та співпраці роблять його вагомим внеском у сферу освіти та організації навчального процесу.

Загалом, розроблений застосунок доводить, що використання ElectronJS та React разом з відкритим кодом та практиками чистого коду може значно вплинути на розробку десктопних застосунків для студентів. Він надає ефективні інструменти для організації та покращення навчального процесу, сприяє спільноті розробників та сприяє розвитку відкритого програмного забезпечення.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Себеста Р.В. Основні поняття мов програмування. Поняття мов програмування. 5-е видання. М.: Вільямс, 2001. 672 р.
2. ReactJS. URL: https://react.dev/ (дата звернення 20.05.2023).
3. Node Package Manager. URL: https://docs.npmjs.com/ (дата звернення 20.05.2023).
4. ElectronJS. URL: https://www.electronjs.org/docs/latest/ (дата звернення 21.05.2023).
5. Git. URL: https://Git-scm.com/doc (дата звернення 01.05.2023).
6. GitHub. URL: https://docs.Github.com/en (дата звернення 11.05.2023).
7. Visual Studio – Вікіпедія. URL: http://ru.wikipedia.org/wiki/Visual_Studio#Visual_Studio_2010 (дата звернення 28.05.2022).
8. Фленаган Д. Книга JavaScript 6 видання. 2012. 1080 с.
9. Мартін Р.С. Чистий код. Створення і рефакторинг 2008. 464 с.

ДОДАТКИ

ДОДАТОК А

Лістинг компоненту Electron.js

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

const { app, BrowserWindow } = require('electron');
const isDev = require('electron-is-dev');

function createWindow() {
  const win = new BrowserWindow({
    width: 800,
    height: 600,
    webPreferences: {
      nodeIntegration: true,
    },
  });

  // and load the index.html of the app.
  // win.loadFile("index.html");
  win.loadURL(
    isDev
      ? 'http://localhost:3000'
      : `file://${path.join(__dirname, '../build/index.html')}`
  );
  // Open the DevTools.
  if (isDev) {
    win.webContents.openDevTools({ mode: 'detach' });
  }
}

app.whenReady().then(createWindow);

app.on('window-all-closed', () => {
    app.quit();
});

app.on('activate', () => {
  if (BrowserWindow.getAllWindows().length === 0) {
    createWindow();
  }
});
```

ДОДАТОК Б

Лістинг компоненту ToDoList.js

```
import React, { useState } from 'react';
import './ToDoList.css';
import Schedule from '../Schedule'
const ToDoList = () => {
  const [tasks, setTasks] = useState([]);
  const [newTask, setNewTask] = useState('');
  const handleInputChange = (event) => {
    setNewTask(event.target.value);
  };
  const addTask = () => {
    if (newTask.trim() !== '') {
      setTasks([...tasks, newTask]);
      setNewTask('');
    }
  };
  const deleteTask = (index) => {
    const updatedTasks = [...tasks];
    updatedTasks.splice(index, 1);
    setTasks(updatedTasks);
  };
  return (
    <div>
      <h1>ToDo List</h1>
      <input type="text" value={newTask} onChange={handleInputChange} />
      <button onClick={addTask}>Add Task</button>
      <ul>
        {tasks.map((task, index) => (
          <li key={index}>
            {task}
            <button onClick={() => deleteTask(index)}>Delete</button>
          </li>
        ))}
      </ul>
      </>
    </div>
  );
};
export default ToDoList;
```

ДОДАТОК В

Лістинг компоненту Schedule.css

```
.schedule {
  display: flex;
  flex-direction: column;
  background-color: #fffacd;
  padding: 20px;
  border-radius: 10px;
}
.time {
  font-family: system-ui;
  position: absolute;
  padding-left: 800px;
  font-size: 30px;
  font-weight: bold;
}
.day {
  margin-bottom: 20px;
  padding: 20px;
  background-color: #c0e1ff;
  border-radius: 5px;
  max-width: 600px;
  width: 500px;
  margin-right: auto;
  font-size: 25 px;
}

h2 {
  font-size: 20px;
  font-weight: bold;
  margin-bottom: 10px;
}

ul {
  font-family: system-ui;

  list-style-type: none;
  padding-left: 0;
  margin-bottom: 0;
}

li {
  font-family: "Acme";
```

```
  border: 3px dotted #ffff00a0;
  border-radius: 5px;
  margin-bottom: 5px;

}
button {
  margin-left: 400px;
  display: inline-block;
  flex-direction: row-reverse;
  padding: 3px 17px;
  font-size: 17px;
  cursor: pointer;
  text-align: center;
  text-decoration: none;
  outline: none;
  color: #fff;
  background-color: #1533db;
  border: none;
  border-radius: 15px;
}
.button:hover {
  background-color: #1e50e6;
}
.button:active {
  background-color: #0008fe;
  box-shadow: 0 2px #666;
  transform: translateY(1px);
}

p {
  margin: 0;
}
```

ДОДАТОК Г

Лістинг компоненту Schedule.js

```
import React from 'react';
import './Schedule.css';

const Schedule = () => {
  return (
    <div className="schedule">


      <div className='time'>
        <ul>
          <li>
            Перша пара : 9:00-10:35
          </li>
          <li>
            Друга пара : 10:45-12:20
          </li>
          <li>Третя пара : 12:45-14:20</li>
        </ul>
      </div>
      <div className="day">
        <h2>Понеділок:</h2>
        <ul>
          <li>1 Нема Пари

          </li>
          <li>2 Нема Пари

          </li>
          <li>3 Електронна комерція
            <button
href='https://us02web.zoom.us/j/3394775925?pwd=ekVmV0V0Z3k1bTFDR3dWOURBYk
ovZz09' class="button">Connect</button>
          </li>
        </ul>
      </div>

      <div className="day">
        <h2>Вівторок:</h2>
        <ul>
          <li>1  Обробка зображень <button
href='https://us02web.zoom.us/j/7863384194?pwd=RUhXLzFOd1NKM0lHaERjci92bk
```

```
5EdzO9' class="button">Connect</button></li>
          <li>2  ГІС <button
href='https://us02web.zoom.us/j/2662039888?pwd=V2dCeGhJNXJueHRidUdib2llME
tNdzO9' class="button">Connect</button></li>
          <li>3 Обробка зображень <button
href='https://us02web.zoom.us/j/7863384194?pwd=RUhXLzFOd1NKMOlHaERjci92bk
5EdzO9' class="button">Connect</button></li>
        </ul>
      </div>

      <div className="day">
        <h2>Середа:</h2>
        <ul>
          <li>1 ТРСПО <button
href='https://us02web.zoom.us/j/3394775925?pwd=ekVmV0V0Z3k1bTFDR3dWOURBYk
ovZzO9' class="button">Connect</button></li>
          <li>2 ГІС <button
href='https://us02web.zoom.us/j/2662039888?pwd=V2dCeGhJNXJueHRidUdib2llME
tNdzO9' class="button">Connect</button></li>
        </ul>
      </div>

      <div className="day">
        <h2>Четверг:</h2>
        <ul>
          <li>1 ТРСПО  <button
href='https://us04web.zoom.us/j/8692470204?pwd=RWxhU2huR0xzc2hEakVIMVF2a1
NXZzO9' class="button">Connect</button></li>
          <li>3 Електронна комерція  <button
href='https://us02web.zoom.us/j/3394775925?pwd=ekVmV0V0Z3k1bTFDR3dWOURBYk
ovZzO9' class="button">Connect</button></li>
        </ul>
      </div>

      <div className="day">
        <h2>Пятница, Суббота, Воскресенье:</h2>
        <p>Нет пар</p>
      </div>
    </div>
  );
};

export default Schedule;
```

ДОДАТОК Д

Лістинг компоненту Schedule.js

```
import React from 'react';
import './Schedule.css';

const Schedule = () => {
  return (
    <div className="schedule">


      <div className='time'>
        <ul>
          <li>
            Перша пара : 9:00-10:35
          </li>
          <li>
            Друга пара : 10:45-12:20
          </li>
          <li>Третя пара : 12:45-14:20</li>
        </ul>
      </div>
      <div className="day">
        <h2>Понеділок:</h2>
        <ul>
          <li>1 Нема Пари

          </li>
          <li>2 Нема Пари

          </li>
          <li>3 Електронна комерція
            <button
href='https://us02web.zoom.us/j/3394775925?pwd=ekVmV0V0Z3k1bTFDR3dWOURBYk
ovZz09' class="button">Connect</button>
          </li>
        </ul>
      </div>

      <div className="day">
        <h2>Вівторок:</h2>
        <ul>
          <li>1  Обробка зображень <button
href='https://us02web.zoom.us/j/7863384194?pwd=RUhXLzF0d1NKM0lHaERjci92bk
```

```
5EdzO9' class="button">Connect</button></li>
          <li>2  ГІС <button
href='https://us02web.zoom.us/j/2662039888?pwd=V2dCeGhJNXJueHRidUdib2llME
tNdzO9' class="button">Connect</button></li>
          <li>3 Обробка зображень <button
href='https://us02web.zoom.us/j/7863384194?pwd=RUhXLzFOd1NKMOlHaERjci92bk
5EdzO9' class="button">Connect</button></li>
        </ul>
      </div>

      <div className="day">
        <h2>Середа:</h2>
        <ul>
          <li>1 ТРСПО <button
href='https://us02web.zoom.us/j/3394775925?pwd=ekVmV0V0Z3k1bTFDR3dWOURBYk
ovZzO9' class="button">Connect</button></li>
          <li>2 ГІС <button
href='https://us02web.zoom.us/j/2662039888?pwd=V2dCeGhJNXJueHRidUdib2llME
tNdzO9' class="button">Connect</button></li>
        </ul>
      </div>

      <div className="day">
        <h2>Четверг:</h2>
        <ul>
          <li>1 ТРСПО  <button
href='https://us04web.zoom.us/j/8692470204?pwd=RWxhU2huR0xzc2hEakVIMVF2a1
NXZzO9' class="button">Connect</button></li>
          <li>3 Електронна комерція  <button
href='https://us02web.zoom.us/j/3394775925?pwd=ekVmV0V0Z3k1bTFDR3dWOURBYk
ovZzO9' class="button">Connect</button></li>
        </ul>
      </div>

      <div className="day">
        <h2>Пятница, Суббота, Воскресенье:</h2>
        <p>Нет пар</p>
      </div>
    </div>
  );
};

export default Schedule;
```

ДОДАТОК Е

Лістинг конфігурацій Package.json

```
{
  "name": "electron-react-diary",
  "version": "0.1.0",
  "main": "public/electron.js",
  "private": true,
  "dependencies": {
    "@testing-library/jest-dom": "^5.16.5",
    "@testing-library/react": "^13.4.0",
    "@testing-library/user-event": "^13.5.0",
    "bootstrap": "^4.0.0-alpha.4",
    "cross-env": "^7.0.3",
    "express": "^4.18.2",
    "mongodb": "^5.5.0",
    "react": "^18.2.0",
    "react-dom": "^18.2.0",
    "react-router-dom": "^6.12.1",
    "react-scripts": "5.0.1",
    "web-vitals": "^2.1.4"
  },
  "scripts": {
    "start": "cross-env BROWSER=none react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject",
    "dev": "concurrently -k \"cross-env BROWSER=none npm start\"
\"npm:electron .\"",
    "electron": "electron .",
    "pack": "electron-builder --dir",
    "dist": "electron-builder"
  },
  "eslintConfig": {
    "extends": [
      "react-app",
      "react-app/jest"
    ]
  },
  "browserslist": {
    "production": [
      ">0.2%",
      "not dead",
      "not op_mini all"
```

```
    ],
    "development": [
      "last 1 chrome version",
      "last 1 firefox version",
      "last 1 safari version"
    ]
  },
  "devDependencies": {
    "concurrently": "^8.0.1",
    "electron": "^24.3.1",
    "electron-builder": "^23.6.0",
    "electron-is-dev": "^2.0.0",
    "wait-on": "^7.0.1"
  },
  "build": {
    "appId": "com.example.myapp",
    "productName": "Diary",
    "directories": {
      "output": "build"
    },
    "win": {
      "icon": "path/to/icon.ico"
    },
    "mac": {
      "icon": "path/to/icon.icns"
    },
    "linux": {
      "icon": "path/to/icon.png"
    }
  }
}
```

ДОДАТОК Є

Лістинг компоненту index.js

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
// import App from './App';
import reportWebVitals from './reportWebVitals';
import { BrowserRouter } from 'react-router-dom';
import ToDoList from './components/ToDoList';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <BrowserRouter>
    <ToDoList/>


    </BrowserRouter>
  </React.StrictMode>
);

reportWebVitals();
```