Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя
(повне найменування вищого навчального закладу)
Факультет прикладних інформаційних технологій та електроінженерії
(назва факультету)
Кафедра автоматизації технологічних процесів та виробництв
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(освітній рівень)

на тему: **Розробка автоматизованої системи моніторингу та складання графіків погоди**

Виконав: студент 4 курсу, групи КАс-41
Спеціальність *151*
“Автоматизазія та комп’ютерно-інтегровані технології”
(шифр і назва спеціальності)

(підпис) Сенко В. Ю. (прізвище та ініціали)

Керівник (підпис) Дмитрів О.Р. (прізвище та ініціали)

Нормоконтроль (підпис) Козбур І.Р. (прізвище та ініціали)

Завідувач кафедри (підпис) Савків В.Б. (прізвище та ініціали)

Рецензент (підпис) Дідич І.С. (прізвище та ініціали)

м. Тернопіль – 2021

АНОТАЦІЯ

Дана кваліфікаційній робота складається з таких розділів:

– вступу, де проводиться опис автоматизації збору даних та створення графіків для спостереження та контролю показників погоди;

– аналітичної частини, в якій проведено аналіз а також порівняння інших готових програмних рішень які використовують інші технології;

– проектної частини, в якій проводиться опис роботи автоматизації збору даних та створення графіків для спостереження та контролю показників погоди;

– спеціальної частини, в якій приведено опис використаних в дипломній роботі бібліотеки та мови програмування;

– безпеки життєдіяльності, основи охорони праці, де подано заходи для забезпечення безпеки роботи з комп'ютером.

This qualifying work consists of the following sections:

- introduction, which describes the automation of data collection and creation of schedules for monitoring and control of weather indicators;

- analytical part, which analyzes and compares other ready-made software solutions that use other technologies;

- project part, which describes the work of automation of data collection and creation of schedules for monitoring and control of weather indicators;

- a special part, which provides a description of the library and programming language used in the thesis;

- life safety, basics of labor protection, where measures are taken to ensure the safety of computer work.

ЗМІСТ

ВСТУП

Тривалий час моніторинг погоди був в основному розвагою любителів-ентузіастів, але за останнє століття він перетворився на добре організовану та професійну глобальну діяльність, що відображає його вирішальне значення для широкого кола економічних, екологічних, цивільних заходів та сільського господарства. Змінні погоди, такі як швидкість та напрямок вітру, температура повітря, вологість та кількість опадів, можуть бути важливими факторами, що визначають хід широкого кола подій. Наприклад, сільське господарство завжди сильно залежало від погоди та прогнозів погоди, як з точки зору контролю якості та кількості врожаю, так і впливу на здатність фермера обробляти землю чи пасти свій запас.

Водні ресурси, як правило, критично залежать не тільки від опадів, але й інших погодних явищ, які в сукупності сприяють росту рослин, фотосинтезу та випаровуванню. Подібно до того, як пилок та розповсюдження насіння в атмосфері майже повністю зумовлені погодою, так само залежить напрямок і відстань руху атмосферного забруднення.

Моніторинг погоди також важливий не тільки для визначення сучасного клімату, але також для виявлення змін клімату та надання даних для введення в моделі, які дозволяють нам передбачати майбутні зміни в нашому середовищі.

Метою роботи є розробка зручного користувацького інтерфейсу для спрощення проведення лабораторних робіт в навчальних закладах чи презентацій. Також розроблене в ході виконання дипломної роботи програмне забезпечення може використовуватися для самостійного моніторингу погоди.

В роботі оцінено потенціал збереження даних на хмарному сховищі та простоту і швидкість побудови графіків погоди за допомогою даної розробки. Це дослідження включало оцінку надійності хмарних сховищ, а також точність побудови графіків за допомогою програмного забезпечення Chart.js призначеного для побудови графіків різного типу та складності. А також перевірена точність даних з різних ресурсів що надають дані щодо погоди.

1 АНАЛІТИЧНА ЧАСТИНА

У цій частині описано способи збору даних, оцінки програмного забезпечення Chart.js та способу аналізу. Методологія розділена на три розділи: Збір даних, Оцінка палетизації Chart.js, аналіз результату виконання програми.

Для початку дослідження літератури з використанням бібліотеки ТНТУ з електронною базою даних послужили базою для збору даних для теоретичних основ. Зібрані дані складалися з різних показників погоди таких як вологість, температура, вологість та інші. Дані складали фільми, фотографії та дані з різних ресурсів моніторингу погоди. Для реалізації користувацького інтерфейсу використовувалась HTML - мова розмітки гіпертекстових документів, CSS - Каскадні таблиці стилів, JavaScript - динамічна прототипна мова програмування.

1.1. Internet

Internet – системна архітектура, яка революціонізувала комунікації та методи комерції, дозволивши різним комп’ютерним мережам у всьому світі взаємозв’язуватися. Іноді його називають "мережею мереж", Інтернет виник у США в 1970-х роках, але не став видимим для широкої громадськості лише на початку 1990-х. До 2020 року, за оцінками, приблизно 4,5 мільярда людей, або більше половини світового населення, мали доступ до Інтернету.

Інтернет надає настільки потужну та загальну можливість, що її можна використовувати майже для будь-яких цілей, що залежать від інформації, і до неї доступний кожен, хто підключається до однієї із складових мереж. Він підтримує спілкування людей через соціальні медіа, електронну пошту (електронну пошту), "чати", групи новин, а також передачу аудіо та відео та дозволяє людям співпрацювати у багатьох різних місцях. Він підтримує доступ до цифрової інформації багатьма програмами, включаючи Всесвітню павутину. Інтернет виявився місцем виникнення великої та зростаючої кількості "електронного бізнесу" (включаючи дочірні компанії традиційних "цегляно-

будівельних" компаній), які здійснюють більшу частину своїх продажів та послуг через Інтернет.

Першими комп’ютерними мережами були спеціальні системи, такі як SABER (система бронювання авіакомпаній) та AUTODIN I (система оборонного управління), розроблені та впроваджені наприкінці 1950-х та на початку 1960-х років. На початку 1960-х виробники комп'ютерів почали використовувати напівпровідникову технологію в комерційних продуктах, і як звичайні системи обробки партій, так і розподіл часу існували у багатьох великих, технологічно просунутих компаніях. Системи розподілу часу дозволили швидко розподілити ресурси комп’ютера з кількома користувачами, проїхавшись по черзі користувачів так швидко, що комп’ютер з’явився присвяченим завданням кожного користувача, незважаючи на існування багатьох інших, які отримують доступ до системи «одночасно». Це призвело до ідеї спільного використання комп'ютерних ресурсів (які називаються хост-комп'ютерами або просто хостами) по всій мережі. Передбачалася взаємодія між хостом та хостом, поряд із доступом до спеціалізованих ресурсів (таких як суперкомп'ютери та системи масового зберігання) та інтерактивний доступ віддалених користувачів до обчислювальних можливостей систем розподілу часу, розташованих в інших місцях. Вперше ці ідеї були реалізовані в ARPANET, яка встановила перше мережеве з'єднання від хосту до хоста 29 жовтня 1969 р. Вона була створена Агентством перспективних дослідницьких проектів (ARPA) Міністерства оборони США. ARPANET була однією з перших комп'ютерних мереж загального призначення. Він з'єднав комп'ютери з розподілом часу на державних науково-дослідних сайтах, головним чином в університетах США, і незабаром він став критично важливим елементом інфраструктури для дослідницького співтовариства з питань інформатики в США. Швидко з'явилися інструменти та програми - такі як простий протокол передачі пошти (SMTP, який зазвичай називають електронною поштою) для надсилання коротких повідомлень та протокол передачі файлів (FTP) для довших передач. Для того, щоб досягти економічно вигідного інтерактивного

зв'язку між комп'ютерами, які, як правило, обмінюються даними з короткими потоками даних, ARPANET застосував нову технологію комутації пакетів. Комутація пакетів приймає великі повідомлення (або фрагменти комп'ютерних даних) і розбиває їх на менші керовані частини (відомі як пакети), які можуть самостійно переміщатися по будь-якій доступній схемі до цільового пункту призначення, де фрагменти збираються повторно. Таким чином, на відміну від традиційного голосового зв'язку, комутація пакетів не вимагає єдиної виділеної схеми між кожною парою користувачів.

1.2. Domain Name System

DNS, в системі повних імен доменів, мережева послуга, яка перетворює між адресами "імен" у Всесвітній павутині та числовими адресами в Інтернеті.

Поняття сервера імен виникло в результаті перших комп'ютерних мереж в середині 1970-х. Кожен комп'ютер у мережі був ідентифікований унікальним номером, але, оскільки розміри комп'ютерних мереж зростали, користувачам було важко відстежувати, яка машина відповідає кожному номеру. Щоб відстежувати, дослідники розробили базу даних, яка перетворила числову адресу кожного комп’ютера в доменне ім’я, що являє собою рядок букв і цифр, які, як правило, легше запам’ятовувати користувачам, ніж числові адреси.

Сучасні DNS-сервери працюють подібним чином із набором баз даних, що працюють на серверах, розкиданих по Інтернету. DNS-сервери використовують ієрархічну структуру для організації доменних імен. Існує два основних типи DNS-серверів: основний, який містить бази даних, і вторинний, який отримує інформацію з первинних баз даних. Основною формою цієї структури є назва машини, за якою слідує домен верхнього рівня (TLD), розділений крапками (крапками). Наприклад, britannica.com має доменне ім'я "britannica" та TLD "com". Найпоширеніший тип TLD є загальним, наприклад, "com", "gov" або "edu", хоча існують також TLD з кодом країни, такі як "uk", "ca" або "au", і спонсорується TLD, такі як подорожі чи робота. Імена доменів та доменів верхнього рівня зареєстровані та контролюються Інтернет-корпорацією з присвоєними номерами та іменами (ICANN).

DNS, який працює над архітектурою протоколу управління передачею / протоколом Інтернету (TCP / IP), швидше за все, буде продовжуватися в найближчому майбутньому як стандарт для доступу до веб-сайтів Інтернету.

1.3. Uniform Resource Identifier

URI (Uniform Resource Identifier) - це компактна послідовність символів, яка ідентифікує абстрактний або фізичний ресурс. Ця специфікація визначає загальний синтаксис URI та процес вирішення посилань на URI, які можуть бути у відносній формі, а також вказівки та міркування щодо використання URI в Інтернеті. Синтаксис URI визначає граматику, яка є набором усіх дійсних URI, дозволяючи реалізації аналізувати загальні компоненти посилання URI, не знаючи специфічних для схеми вимог кожного можливого ідентифікатора. Ця специфікація не визначає генеративної граматики для URI; це завдання виконується індивідуальними специфікаціями кожної схеми URI.

Уніфікований ідентифікатор ресурсу (URI) забезпечує простий і розширюваний спосіб ідентифікації ресурсу. Ця специфікація синтаксису та семантики URI походить від концепцій, запроваджених глобальною інформаційною ініціативою Всесвітньої павутини, використання цих ідентифікаторів яких походить з 1990 року та описано в "Універсальних ідентифікаторах ресурсів у WWW" [RFC1630]. Синтаксис розроблений з урахуванням рекомендацій, викладених у "Функціональних рекомендаціях для локаторів Інтернет-ресурсів" [RFC1736] та "Функціональні вимоги до єдиних назв ресурсів" [RFC1737].

1.4. Uniform Resource Locator

URL-адреса в повному уніфікованому локаторі ресурсів, адреса ресурсу в Інтернеті або файлу, що зберігається локально. Ресурсом може бути будь-який тип файлу, що зберігається на сервері, наприклад, веб-сторінка, текстовий файл, графічний файл або прикладна програма. Адреса містить три елементи: тип протоколу, який використовується для доступу до файлу (наприклад, HTTP для веб-сторінки, ftp для сайту FTP); доменне ім'я або IP-адреса сервера, де

знаходиться файл; і, за бажанням, ім’я шляху до файлу (тобто опис розташування файлу). Наприклад, URL-адреса http://www.britannica.com/heritage вказує браузеру використовувати протокол HTTP, перейти на веб-сервер www.britannica.com та отримати доступ до файлу з назвою спадщина. Схема URL-адреси та її частин зображено на рисунку 1.1.

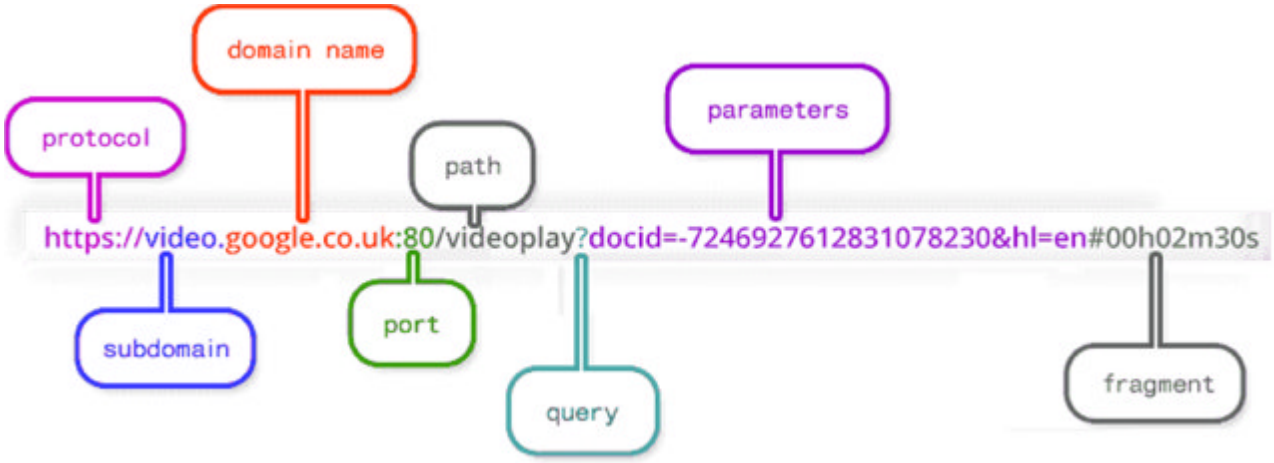


Рисунок 1.1 – Схема URL-адреси

1.5. Мережева модель OSI

Довідкова модель Open Systems Interconnect (OSI) є основою комерційно доступних архітектур мережевих служб. Інші мережеві протоколи, розроблені самостійно, вільно відповідають моделі. Прикладом є набір протоколів Інтернету TCP / IP.

Довідкова модель OSI є зручним каркасом для мережевих концепцій. В основному дані вводяться в мережу відправником. Дані передаються по комунікаційному з'єднанню і доставляються до приймача. Для цього різноманітне мережеве обладнання та програмне забезпечення повинні працювати разом.

Довідкова модель OSI ділить функції мережевого зв'язку на сім рівнів.

Кожен рівень протоколу виконує послуги для рівня над ним. Визначення ISO рівнів протоколу надає дизайнерам певну свободу реалізації. Наприклад, деякі програми пропускають рівні презентації та сеансу для безпосереднього взаємодії з транспортним рівнем.

Рівень 1: Фізичний рівень - апаратний рівень моделі. У системах SPARC (TM) він складається з роз’єму до мережевого середовища передачі, будь-яких мультиплексорних коробок та кабелів.

Рівень 2: Рівень передачі даних - чи є відправлення та отримання. На кінці відправлення програмне забезпечення Ethernet (або подібне) упорядковує дані у пакети відповідного розміру та упаковує їх. Упаковка містить фізичну адресу передбачуваного приймача. Шар також передає пакети повідомлень і повторно їх передає, якщо це необхідно. На приймальному кінці обладнання Ethernet розпізнає пакети зі своєю адресою та приймає їх. Програмне забезпечення Ethernet знімає упаковку для передачі та збирає дані. Він може виявити помилки передачі.

Рівень 3: Мережевий рівень - чи здійснюється маршрутизація повідомлень, включаючи переклад з логічної на фізичну адресу. Інтернет-протокол (IP) є звичайним мережевим рівнем для систем SPARC.

Рівень 4: Транспортний рівень - керує потоком даних у мережі. У SunOS 5.8 може використовуватися будь-який з інтерфейсів транспортного рівня (TLI), протоколу управління передачею (TCP) або протоколу користувацьких датаграм (UDP). У системах SPARC служба режиму підключення зазвичай надається через TCP, а послуга без підключення - через UDP.

Рівень 5: Рівень сеансу - керує надійними сеансами між процесами. Віддалені виклики процедур (RPC) належать цьому шару. Інтерфейс на цьому рівні дозволяє віддалене спілкування за допомогою семантики викликів функцій.

Рівень 6: Презентаційний рівень - виконує переклад між місцевим представленням даних на комп'ютері та незалежним від процесора форматом, який надсилається через мережу. У середовищі SunOS 5.8 незалежним від процесора форматом даних є XDR.

Рівень 7: Рівень програми - на цьому верхньому рівні знаходяться програми та послуги на рівні користувача. Прикладами програм на рівні

користувача є telnet, rlogin, ftp та yppasswd. Прикладами послуг є NFSTM, NIS та DNS.

Галузеві стандарти були визначені або визначаються для кожного рівня еталонної моделі. Для кожного рівня визначено два стандарти: один визначає інтерфейс до послуг, що надаються шаром, а інший - протокол, який спостерігаються службами на рівні. Користувачі стандарту службового інтерфейсу повинні мати можливість ігнорувати протокол та будь-які інші деталі реалізації рівня.

1.6. TCP/IP

TCP / IP, в повному обсязі Протокол управління передачею / Інтернет-протокол, стандартні протоколи Інтернет-зв'язку, які дозволяють цифровим комп’ютерам спілкуватися на великі відстані. Інтернет - це мережа з комутацією пакетів, в якій інформація розбивається на невеликі пакети, що надсилаються одночасно по безлічі різних маршрутів, а потім знову збираються на кінці, що приймає. TCP - це компонент, який збирає та збирає пакети даних, тоді як IP відповідає за те, щоб пакети були відправлені в потрібний пункт призначення. TCP / IP був розроблений в 1970-х і прийнятий як стандарт протоколу для ARPANET (попередника Інтернету) в 1983 році.

1.7. Hyper Text Transfer Protocol

HTTP(HyperText Transfer Protocol) Протокол зв'язку, що використовується для підключення до веб-серверів в Інтернеті або в локальній мережі (інтрамережі). Основна функція HTTP - встановити зв’язок із сервером та відправити HTML-сторінки назад у браузер користувача. Він також використовується для завантаження даних із сервера або в браузер, або в будь-яку програму-запит, яка використовує HTTP.

Адреси веб-сайтів починаються з префіксу http: //; однак веб-браузери зазвичай використовують протокол HTTP за замовчуванням. Наприклад, набирати www.yahoo.com - те саме, що набирати http://www.yahoo.com. Насправді потрібно вводити лише yahoo.com. Решту додає браузер. HTTPS HTTPS (HTTP Secure) - це зашифрована версія HTTP (див. HTTPS). HTTP-з'єднання без громадянства - це система запитів / відповідей без громадянства.

Зв'язок між клієнтом та сервером підтримується лише для негайного запиту, і з'єднання закрито. Після того, як клієнт HTTP встановлює TCP-з'єднання з сервером і надсилає йому команду запиту, сервер відправляє назад свою відповідь і закриває з'єднання. Перша версія HTTP спричинила значні накладні витрати. Кожного разу, коли надсилався запит на графічний файл на сторінці, між браузером та сервером потрібно було встановити нове з’єднання з протоколом.

Коли ми хочемо переглядати веб-сторінки, ми можемо використовувати багато типів комп’ютерів (наприклад, ноутбуки, настільні комп’ютери та телефони), якщо на комп’ютері встановлена програма для браузера.

Користувач або вводить уніфікований локатор ресурсів (URL) у браузері, або переходить за посиланням із вже відкритої сторінки. Зауважте щось про цю URL-адресу: вона починається з "http". Це сигнал для браузера, що йому потрібно використовувати HTTP для отримання документа для цієї URL-адреси.

Зазвичай ми вводимо в браузери приємні для людини URL-адреси, наприклад "khanacademy.org" та "wikipedia.org". Ці доменні імена

відображають IP-адреси, і справжнє розташування комп’ютерів домену. Цим займається система доменних імен. Браузер використовує DNS для зіставлення домену з IP-адресою.

Як тільки браузер ідентифікує IP-адресу комп’ютера, на якому розміщується запитувана URL-адреса, він надсилає запит HTTP. Перше слово - це HTTP: "GET". Є й інші слова для інших дій в Інтернеті, наприклад, надсилання даних форми ("POST").

Наступна частина визначає шлях: "/index.html". Хост-комп’ютер зберігає вміст всього веб-сайту, тому браузер повинен чітко визначити, яку сторінку завантажувати.

Заключна частина першого рядка визначає протокол і версію протоколу: "HTTP / 1.1". Другий рядок визначає домен запитуваної URL-адреси. Це корисно, якщо головний комп’ютер зберігає вміст для кількох веб-сайтів. Хост відправляє відповідь HTTP. Після того, як хост-комп'ютер отримує запит HTTP, він надсилає відповідь із вмістом і метаданими про нього.

Відповідь починається з протоколу та версії "HTTP / 1.1". Наступне число - це дуже важливий код стану HTTP, а в даному випадку це 200. Цей код означає успішне отримання документ.

Якщо серверу не вдалося отримати документ, коди стану надають додаткову інформацію, наприклад, якщо помилка сталася через помилку користувача або помилку сервера. Наприклад, найвідоміший код стану - 404 ("Файл не знайдено"). Це трапляється кожного разу, коли ви відвідуєте шлях на сервері, який не відповідає жодному документу.

Оскільки користувачі мають звичку неправильно вводити URL-адреси, 404 трапляються часто, тому веб-сайти часто отримують задоволення від 404 веб-сторінок.

Наступною частиною відповіді HTTP є заголовки. Вони надають браузеру додаткові деталі та допомагають браузеру відтворити вміст. Тип вмісту повідомляє браузеру, який тип документа він відправляє назад. Поширений тип в Інтернеті - "text / html", оскільки всі веб-сторінки є текстовими файлами

HTML. Можливі й інші типи, такі як зображення ("image / png"), відео ("video / mpeg"), сценарій ("application / javascript") та все інше, що ви можете завантажити у свій браузер. Довжина вмісту визначає довжину документа в байтах, що допомагає браузеру дізнатися, скільки часу потрібно для завантаження файлу. Нарешті, відповідь HTTP виписує фактичний запитуваний документ. Ця сторінка є простим файлом HTML. Браузер надає відповідь. Тепер браузер має всю інформацію, необхідну для надання запитуваного документа.

У HTTP версії 1.1 кілька файлів можна завантажувати за одного підключення. Це також покращило кешування та спростило створення віртуальних хостів (кілька веб-сайтів на одному сервері). Див. HTTP / 2, заголовок HTTP та файл cookie.

1.9. Веб-браузер

Вуб-браузер– програмне забезпечення, яке дозволяє користувачеві комп’ютера знаходити та переглядати інформацію в Інтернеті. Веб-браузери інтерпретують теги HTML у завантажених документах та форматують відображувані дані відповідно до набору стандартних правил стилю.

Коли британський вчений Тім Бернерс-Лі винайшов Всесвітню павутину, він також створив перший браузер WorldWideWeb, який став доступним у 1991 році і також міг використовуватися для редагування веб-сторінок. Використання Інтернету швидко розширилося після виходу Mosaic у 1993 році, який використовував графічні маніпуляції "вказуй і клацни" і був першим браузером, який відображав як текст, так і зображення на одній сторінці. Команда, яка стоїть за Mosaic, створила Netscape Navigator, яка була оптимізована для домашніх користувачів, які переглядали на повільних швидкостях комутованих модемів. Netscape Navigator став домінуючим веб-браузером незабаром після його випуску в 1994 році. Перший браузер з вкладками, в якому користувач міг відвідувати інший веб-сайт, не відкриваючи абсолютно нового вікна, дебютував того ж року. Microsoft випустила свій браузер Internet Explorer в 1995 році. Internet Explorer постачався в комплекті з

операційною системою Microsoft Windows і замінив Netscape Navigator як домінуючий браузер наприкінці 1990-х.

Safari від Apple був випущений у 2003 році як браузер за замовчуванням на персональних комп’ютерах Macintosh, а пізніше на iPhone (2007) та iPad (2010). Safari 2.0 (2005) був першим браузером із режимом конфіденційності, приватним переглядом, у якому програма не зберігала веб-сайти в своїй історії, завантажені файли в кеш-пам’яті та особисту інформацію, введену на веб-сторінках.

Першим серйозним викликом домінуванню Internet Explorer став Firefox від Mozilla, випущений у 2004 році і призначений для вирішення проблем зі швидкістю та безпекою, які мучили Internet Explorer. У 2008 році Google запустив Chrome, перший браузер із ізольованими вкладками, що означало, що при збої однієї вкладки інші вкладки та весь браузер все одно функціонуватимуть. До 2013 року Chrome став домінуючим браузером, перевершивши за популярністю Internet Explorer і Firefox. Microsoft припинила роботу Internet Explorer і замінила його на Edge у 2015 році.

1.8. HyperText Markup Language

HTML (HyperText Markup Language) – система форматування для відображення матеріалів, отриманих через Інтернет. Кожна одиниця пошуку відома як веб-сторінка (з Інтернету), і такі сторінки часто містять гіпертекстові посилання, що дозволяють отримувати пов'язані сторінки. HTML - це мова розмітки для кодування веб-сторінок. Він був розроблений британським вченим сером Тімом Бернерс-Лі в лабораторії ядерної фізики ЦЕРН у Швейцарії протягом 1980-х років. Теги розмітки HTML визначають такі елементи документа, як заголовки, абзаци та таблиці. Вони розмічають документ для показу комп’ютерною програмою, відомою як веб-браузер. Браузер інтерпретує теги, відображаючи заголовки, абзаци та таблиці в макеті, адаптованому до розміру екрана та доступних йому шрифтів.

Документи HTML також містять прив'язки, які є тегами, що визначають посилання на інші веб-сторінки. Якір має вигляд <A HREF= “http://www.britannica.com">> Encyclopædia Britannica </A>, де цитований рядок - це URL (універсальний локатор ресурсів), на який вказує посилання (адреса в Інтернеті) ”), А текст після нього - це те, що з’являється у веб-браузері, підкреслено, щоб показати, що це посилання на іншу сторінку. Те, що відображається як одна сторінка, також може бути сформовано з декількох URL-адрес, деякі містять текст, а інші графіку.

1.9. Cascading Style Sheets

CSS (Cascading Style Sheets) - Каскадні таблиці стилів використовуються для форматування макета веб-сторінок. Їх можна використовувати для визначення стилів тексту, розмірів таблиць та інших аспектів веб-сторінок, які раніше можна було визначити лише в HTML сторінки.

CSS допомагає веб-розробникам створити єдиний вигляд кількох сторінок веб-сайту. Замість того, щоб визначати стиль кожної таблиці та кожного блоку тексту в HTML-коді сторінки, загальновживані стилі потрібно визначати лише один раз у документі CSS. Після того, як стиль визначено в каскадній таблиці стилів, він може використовуватися будь-якою сторінкою, яка посилається на файл CSS. Крім того, CSS дозволяє легко змінювати стилі на декількох сторінках одночасно. Наприклад, веб-розробник може захотіти збільшити розмір тексту за замовчуванням з 10 до 12 пунктів для п’ятдесяти сторінок веб-сайту. Якщо всі сторінки посилаються на одну таблицю стилів, розмір тексту потрібно лише змінити на таблиці стилів, і всі сторінки відображатимуть більший текст.

Хоча CSS чудово підходить для створення стилів тексту, він корисний і для форматування інших аспектів розмітки веб-сторінки. Наприклад, CSS може бути використаний для визначення заповнення комірок комірок таблиці, стилю, товщини та кольору межі таблиці, а також заповнення навколо зображень чи інших об’єктів. CSS дає веб-розробникам точніший контроль над

тим, як виглядатимуть веб-сторінки, ніж HTML. Ось чому сьогодні більшість веб-сторінок містять каскадні таблиці стилів.

1.10. JavaScript

JavaScript - це мова програмування, яка зазвичай використовується при веб-розробці. Спочатку він був розроблений Netscape як засіб для додавання динамічних та інтерактивних елементів до веб-сайтів. Хоча на JavaScript впливає Java, синтаксис більше схожий на C і базується на ECMAScript, мові сценаріїв, розробленій Sun Microsystems.

JavaScript є мовою сценаріїв на стороні клієнта, що означає, що вихідний код обробляється веб-браузером клієнта, а не веб-сервером. Це означає, що функції JavaScript можуть працювати після завантаження веб-сторінки без зв’язку із сервером. Наприклад, функція JavaScript може перевірити веб-форму перед її поданням, щоб переконатися, що всі обов’язкові поля заповнені. Код JavaScript може видавати повідомлення про помилку до того, як будь-яка інформація фактично передається на сервер.

Як і мови сценаріїв на стороні сервера, такі як PHP та ASP, код JavaScript можна вставити в будь-яке місце в HTML веб-сторінки. Однак у HTML відображається лише вихідний код коду на стороні сервера, тоді як код JavaScript залишається повністю видимим у джерелі веб-сторінки. На нього також можна вказати окремий файл .JS, який також можна переглянути в браузері.

Нижче наведено приклад базової функції JavaScript, яка додає два числа. Функція викликається з параметрами 7 і 11. Якби код нижче був включений в HTML веб-сторінки, він би відображав текст "18" у вікні попередження.

Функції JavaScript можна викликати в тегах <script> або коли відбуваються певні події. Приклади включають onClick, onMouseDown, onMouseUp, onKeyDown, onKeyUp, onFocus, onBlur, onSubmit та багато інших. Хоча стандартний JavaScript все ще використовується для виконання основних

функцій на стороні клієнта, зараз багато веб-розробники воліють використовувати бібліотеки JavaScript, такі як jQuery, для додавання більш вдосконалених динамічних елементів на веб-сайти.

1.11. Document Object Model

Об'єктна модель документа (DOM) - це інтерфейс програмування для документів HTML та XML (розширювана мова розмітки). Він визначає логічну структуру документів та спосіб доступу та маніпулювання ним. Вона називається логічною структурою, оскільки DOM не вказує ніяких зв'язків між об'єктами.

DOM - це спосіб представити веб-сторінку в структурованому ієрархічному порядку, щоб програмістам та користувачам було простіше ковзати по документу. За допомогою DOM ми можемо легко отримувати доступ до тегів, ідентифікаторів, класів, атрибутів або елементів та керувати ними за допомогою команд або методів, наданих об’єктом Document.

Структура DOM:

DOM можна сприймати як Дерево або Ліс (більше одного дерева). Термін модель структури іноді використовується для опису деревоподібного подання документа. Однією з важливих властивостей структурних моделей DOM є структурний ізоморфізм: якщо будь-які дві реалізації DOM використовуються для створення представлення одного і того ж документа, вони створять однакову структурну модель із точно однаковими об'єктами та зв'язками.

Документи моделюються за допомогою об'єктів, і модель включає не тільки структуру документа, але також поведінку документа та об'єкти, з яких він складається з подібних елементів тегів з атрибутами в HTML.

Властивості об’єкта документа, до яких об’єкт документа може отримати доступ та зміни.

Об'єкт вікна: Об'єкт вікна завжди знаходиться на вершині ієрархії.

Об'єкт документа: Коли документ HTML завантажується у вікно, він стає об'єктом документа.

Об'єкт форми: представлений тегами форми.

Об'єкти посилання: представлений тегами посилань.

Якірні об’єкти: Він представлений тегами href.

Елементи керування формою :: Форма може мати багато елементів управління, таких як текстові поля, кнопки, перемикачі та прапорці тощо.

write (“рядок”): записує заданий рядок у документ.

getElementById (): повертає елемент, що має вказане значення id.

getElementsByName (): повертає всі елементи, що мають вказане значення імені.

getElementsByTagName (): повертає всі елементи, що мають вказане ім'я тегу.

getElementsByClassName (): повертає всі елементи, що мають вказане ім'я класу.

1.12. Asynchronous JavaScript And XML

AJAX (Asynchronous JavaScript And XML) – це техніка Javascript, яка забезпечує більше інтерактивних інтерфейсів та швидшу продуктивність для адаптивного HTML-вмісту в Інтернеті. AJAX - це відкритий стандарт, який працює в координації з HTML, CSS та Javascript вздовж бічного XML. Він виконується в рамках Javascript з даними, отриманими та збереженими у форматі XML. Прикладом використання AJAX є доступ до даних із зовнішніх джерел, навіть після завершення завантаження веб-сторінки.

Процес, що лежить в основі методології AJAX:

Дані зчитуються з веб-сервера після завантаження сторінки HTML

Дані можна оновлювати, не вимагаючи перезавантаження веб-сторінки

Дані можуть надсилатися на веб-сервер під час роботи у фоновому режимі

Ці кроки є асинхронними та підтримують інтуїтивну та природну взаємодію користувача з веб-сторінками. Він також незалежний від веб-

сервера тим, що він може виконувати завдання в будь-який момент часу, роблячи його керованим даними, а не сторінковим середовищем.

Об'єкт XMLHttpRequest

Далі наведено приклад того, як AJAX може змінювати текст, не вимагаючи від користувача перезавантаження веб-сторінки. Це відбувається через об’єкт XMLHttpRequest, який надсилає запит веб-серверу, а потім відображає результат у Javascript DOM (об’єктна модель документа). Це вбудовано в HTML-сторінку.

Функція, що називається loadData (), отримає вміст файлу, що називається “data.txt”, із веб-сервера. Потім він відображає вміст через DOM. Наступна схема ілюструє процес.

Повертаючись до коду, є умова, яка повинна бути виконана. This.readystate == 4 означає «завершено та отримано відповідь», а this.status == 200 означає, що відповідь веб-сервера має бути HTTP 200 (стан: OK). Якщо сервер не відповів через збій мережі або простої системи, статус HTTP 200 не повертається. Це може бути 500 (внутрішня помилка), 400 (поганий запит) або 404 (запит не знайдено).

Функція отримання

У цьому наступному прикладі ми використовуємо функцію вибору, яка дозволяє відправляти HTTP-запит на веб-сервер. Це повертає обіцянку в Javascript або процес, що очікує на розгляд, доки її не вдасться виконати. Дані обробляються не відразу, а оброблятимуться із зворотного боку веб-сервером. Коли є результат, веб-сервер тоді надішле виконаний законний запит або відхилить запит у разі зловмисної атаки. У операторі обіцянки використовується атрибут .then, який описує, яку дію вжити, коли відповідь отримується від веб-сервера.

Ми збираємось використовувати API для перетворення вартості валюти з обміну грошей. За умови наявності ключа доступу до API можна отримати відповідь від сервера. Якщо ключа доступу до API немає, запит буде відхилено.

1.13. Node Package Manager

npm (Node Package Manager) – це менеджер пакунків для мови програмування JavaScript. npm - це дві речі: по-перше, це онлайн-сховище для публікації проектів з відкритим кодом Node.js; по-друге, це утиліта командного рядка для взаємодії із згаданим сховищем, яка допомагає при встановленні пакунків, керуванні версіями та управлінні залежностями. Безліч бібліотек і програм Node.js публікується в npm, і щодня додається багато іншого. Ці програми можна шукати на https://www.npmjs.com/. Отримавши пакет, який потрібно встановити, його можна встановити за допомогою однієї команди командного рядка.

Скажімо, одного дня ви важко працюєте, розробляючи наступний чудовий додаток. Ви стикаєтесь із проблемою і вирішуєте, що настав час використовувати ту класну бібліотеку, про яку ви постійно чуєте - давайте використаємо асинхронізацію Каолана МакМахона як приклад. На щастя, npm дуже простий у використанні: вам потрібно лише запустити npm install async, і вказаний модуль буде встановлений у поточному каталозі під ./node_modules/. Після встановлення у папку node_modules ви зможете використовувати require () на них так само, як вони були вбудовані.

Приклад глобальної установки - скажімо, кава-скрипт. Команда npm проста: npm install coffee-script -g. Зазвичай це встановлює програму та додає до неї символічне посилання в / usr / local / bin /. Потім це дозволить вам запускати програму з консолі, як і будь-який інший інструмент CLI. У цьому випадку запущена кава тепер дозволить вам використовувати кавовий сценарій REPL.

Іншим важливим використанням npm є управління залежностями. Коли у вас є проект вузла з файлом package.json, ви можете запустити npm install з кореня проекту, і npm встановить усі залежності, перелічені в package.json. Це значно полегшує встановлення проекту Node.js з git-репо! Наприклад, обітниці, тестовий фреймворк Node.js, можна встановити з git, і його одинарна залежність, очі, може бути автоматично оброблена

1.14. Yarn

Yarn - швидкий, надійний та безпечний менеджер залежностей. Створений як альернатива для npm-клієнта, який має деякі пробелми зі швидкістю встановлення пакунків а також з версіями на різних машинах.

1.15. React

React - це бібліотека JavaScript для побудови користувальницьких інтерфейсів. Це рівень перегляду веб-додатків.

В основі всіх програм React лежать компоненти. Компонент - це автономний модуль, який надає певний результат. Ми можемо писати елементи інтерфейсу як кнопку або поле введення як компонент React. Компоненти можна скласти. Компонент може включати один або кілька інших компонентів у свій результат.

Загалом кажучи, для написання програм React ми пишемо компоненти React, які відповідають різним елементам інтерфейсу. Потім ми організуємо ці компоненти всередині компонентів вищого рівня, які визначають структуру нашого додатку.

Наприклад, візьміть форму. Форма може складатися з багатьох елементів інтерфейсу, таких як поля введення, мітки або кнопки. Кожен елемент всередині форми може бути записаний як компонент React. Потім ми писали б компонент вищого рівня, сам компонент форми. Компонент форми визначає структуру форми та включає кожен з цих елементів інтерфейсу всередині неї.

Що важливо, кожен компонент у програмі React дотримується суворих принципів управління даними. Складні, інтерактивні користувальницькі інтерфейси часто включають складні дані та стан додатків. Площа поверхні React обмежена і спрямована на те, щоб дати нам інструменти, які дозволять передбачити, як буде виглядати наш додаток за певних обставин. Ми вивчимо ці принципи пізніше в курсі.

1.16. Redux

Redux - це передбачуваний контейнер стану для програм JavaScript.

Це допомагає писати програми, які поводяться послідовно, працюють в різних середовищах (клієнтських, серверних та власних) і легко тестуються. Крім того, це забезпечує чудовий досвід для розробників, такий як редагування коду в реальному часі в поєднанні з налагоджувачем часу.

Ви можете використовувати Redux разом з React або з будь-якою іншою бібліотекою перегляду. Він крихітний (2 кБ, включаючи залежності), але має велику екосистему додатків.

Redux Toolkit - це наш офіційний рекомендований підхід до написання логіки Redux. Він охоплює ядро Redux і містить пакети та функції, які, на наш погляд, є важливими для створення програми Redux. Набір інструментів Redux використовує запропоновані нами найкращі практики, спрощує більшість завдань Redux, запобігає типовим помилкам та полегшує написання програм Redux.

RTK включає утиліти, які допомагають спростити багато типових випадків використання, включаючи налаштування магазину, створення редукторів та написання незмінної логіки оновлення та навіть створення цілих "фрагментів" стану одночасно.

Незалежно від того, чи є ви новим користувачем Redux, який створює свій перший проект, або досвідченим користувачем, який хоче спростити існуючу програму, Redux Toolkit може допомогти вам покращити ваш код Redux.

Весь глобальний стан вашого додатка зберігається в дереві об’єктів в одному магазині. Єдиний спосіб змінити дерево стану - це створити дію, об’єкт, що описує те, що сталося, і відправити його в магазин. Щоб вказати, як стан оновлюється у відповідь на дію, ви пишете чисті функції редуктора, які обчислюють новий стан на основі старого стану та дії.

1.17. Функціональне програмування

Функціональне програмування стало справді гарячою темою у світі JavaScript. Кілька років тому мало хто з програмістів JavaScript навіть знав, що таке функціональне програмування, але кожна велика кодова програма, яку я бачив за останні 3 роки, активно використовує ідеї функціонального програмування.

Функціональне програмування (часто скорочене FP) - це процес побудови програмного забезпечення шляхом складання чистих функцій, уникаючи спільного стану, змінних даних та побічних ефектів. Функціональне програмування є декларативним, а не імперативним, а стан програми протікає через чисті функції. На відміну від об'єктно-орієнтованого програмування, де стан додатків, як правило, ділиться і розміщується методами в об'єктах.

Функціональне програмування - це парадигма програмування, що означає, що це спосіб думати про побудову програмного забезпечення, заснований на деяких фундаментальних, визначальних принципах (перелічених вище). Інші приклади парадигм програмування включають об'єктно-орієнтоване програмування та процедурне програмування.

Функціональний код має тенденцію бути більш стислим, більш передбачуваним і простішим для перевірки, ніж імперативний або об'єктно-орієнтований код, але якщо ви не знайомі з ним та загальними шаблонами, пов'язаними з ним, функціональний код також може здатися набагато щільнішим, і відповідна література може бути непроникною для новачків.

Якщо ви почнете гуглити умови функціонального програмування, ви швидко зіткнетеся з цегляною стіною академічного жаргону, що може бути дуже лякаючим для початківців. Сказати, що він має криву навчання - це серйозне заниження. Але якщо ви вже деякий час програмуєте на JavaScript, велика ймовірність того, що ви використовували багато функціональних концепцій програмування та утиліт у своєму реальному програмному забезпеченні.

Найскладніше - обернути голову навколо всієї незнайомої лексики. У невинно виглядаючому визначенні є багато ідей, над якими потрібно все

зрозуміти, перш ніж ви зможете зрозуміти сенс функціонального програмування:

- Чисті функції
- Склад функцій
- Уникайте спільного стану
- Уникайте мутуючих станів
- Уникайте побічних ефектів

Іншими словами, якщо ви хочете знати, що означає функціональне програмування на практиці, вам слід почати з розуміння цих основних понять. Чиста функція - це функція, яка: враховуючи однакові входи, завжди повертає однакові результати, і не має побічних ефектів, чисті функції мають безліч властивостей, важливих для функціонального програмування, включаючи посилальну прозорість (ви можете замінити виклик функції отриманим значенням, не змінюючи значення програми).

Композиція функцій - це процес комбінування двох або більше функцій з метою створення нової функції або виконання деяких обчислень. Наприклад, композиція f. g (крапка означає "складене з") еквівалентно f (g (x)) у JavaScript. Розуміння складу функцій є важливим кроком до розуміння того, як побудовано програмне забезпечення за допомогою функціонального програмування.

1.18. Git

Git - це розподілена система контролю версій з відкритим кодом. Тепер це багато слів для визначення Git. Система управління: Це в основному означає, що Git є інструментом відстеження вмісту. Тож Git можна використовувати для зберігання вмісту - він здебільшого використовується для зберігання коду завдяки іншим функціям, які він надає.

Система контролю версій: код, який зберігається в Git, постійно змінюється при додаванні нових кодів. Також багато розробників можуть

додавати код паралельно. Тож Система контролю версій допомагає вирішити цю проблему, зберігаючи історію змін. Крім того, Git надає такі функції, як гілки та злиття, про які я розповім пізніше.

Розподілена система контролю версій: Git має віддалене сховище, яке зберігається на сервері, та локальне сховище, яке зберігається на комп'ютері кожного розробника. Це означає, що код зберігається не просто на центральному сервері, а повна копія коду присутня на всіх комп’ютерах розробників. Git - це розподілена система контролю версій, оскільки код присутній на комп'ютері кожного розробника. Поясню поняття віддаленого та локального сховищ далі в цій статті.

Навіщо потрібна система контролю версій, така як Git. Проекти реального життя, як правило, мають декількох розробників, які працюють паралельно. Тож система контролю версій, така як Git, потрібна, щоб у розробників не було конфліктів коду

Крім того, вимоги в таких проектах часто змінюються. Таким чином, система контролю версій дозволяє розробникам повернутися до попередньої версії коду та повернутися до нього.

Нарешті, іноді кілька проектів, які виконуються паралельно, включають одну і ту ж базу коду. У такому випадку дуже важлива концепція розгалуження в Git.

2 ПРОЕКТНА ЧАСТИНА

2.1. Створення проекту

Найкращим інструментом, який допоможе практично без будь-яких зусиль створити каскад для React-додатку є CLI-інструмент create-react-app. Ми можемо встановити його глобально за допомогою yarn (Рисунок 2.1).

```
➜  ~ yarn global add create-react-app
```

Рисунок 2.1 – Встановлення create-react-app

Для того, щоб створити каскад додатку, за допомогою create-react-app, потрібно скористатись командою create-react-app <назва додатку>, що ми і зробили на рисунку 2.2.

```
➜  dev create-react-app area-photos
```

Рисунок 2.2 – Створення каскаду додатку

Щоб перевірити, чи все пройшло успішно, потрібно спробувати запустити додаток (рисунок 2.3).

```
➜  dev cd ./area-photos
➜  area-photos yarn start
```

Рисунок 2.3 – Тестовий запуск проекту

Якщо все пройшло успішно, то додаток повинен автоматично відкритись у головному браузері на адресі localhost:3000, в чому ми і переконуємось (рисунок 2.4).

Рисунок 2.4 – Вигляд новоствореного додатку

Тепер ми маємо все готове для того, щоб почати розробляти додаток, а саме:

• Якісно налаштований Webpack, який дуже швидко компілює додаток, також з налаштованою “гарячою заміною” коду, та автоматичною перезагрузкою сторінки, що значно пришвидшує процес розробки, та покращує досвід розробника.

• Налаштоване тестування за допомогою Jest

• Створений Git-репозиторій та .gitignore файл (рисунок 2.5)

• Створений package.json файл з відповідними командами та пакетами react і react-dom в залежностях (рисунок 2.6).

```
# See https://help.github.com/ignore-files/ for more about ignoring files.

# dependencies
/node_modules

# testing
/coverage

# production
/build

# misc
.DS_Store
.env.local
.env.development.local
.env.test.local
.env.production.local

npm-debug.log*
yarn-debug.log*
yarn-error.log*
```

2.5 – Вміст файлу .gitignore

```
{
  "name": "area-photos",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "react": "^16.4.0",
    "react-dom": "^16.4.0",
    "react-scripts": "1.1.4"
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test --env=jsdom",
    "eject": "react-scripts eject"
  }
}
```

Рисунок 2.6 – вміст файлу package.json

Щоб перевірити, чи працює автоматичне перезавантаження, спробуємо відредагувати файл App.js в директорії src. Як видно із рисунка 2.6, все працює.

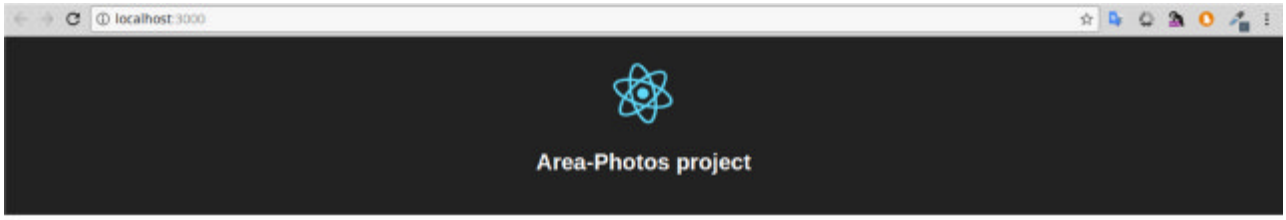


Рисунок 2.6 – Вигляд додатку після невеликої зміни

Щоб перевірити, чи тестування працює правильно, потрібно виконати команду, показану на рисунку 2.7.

2.7 – Запуск тестування

Як бачимо, стандартний тест проходить успішно (рисунок 2.8)

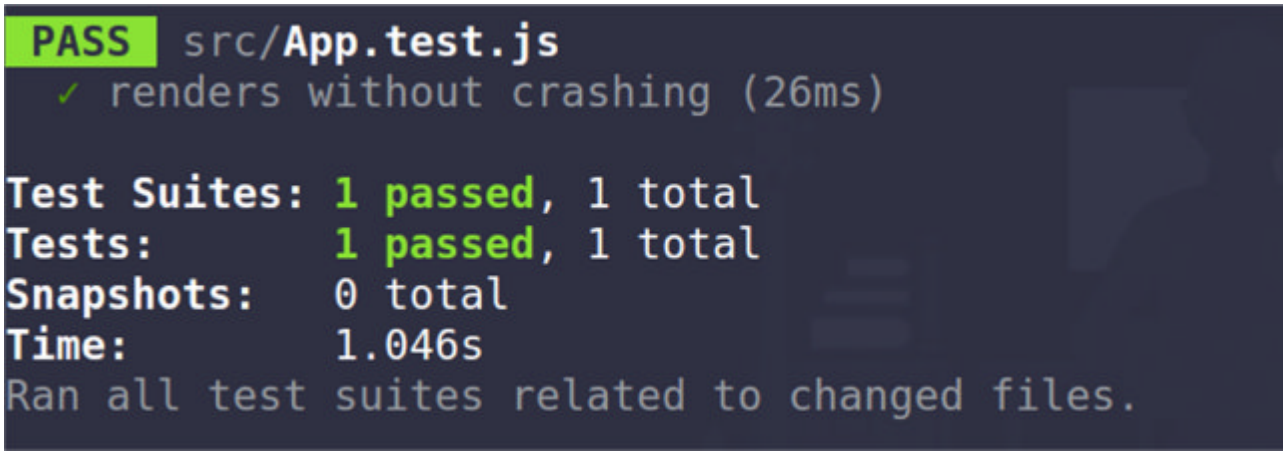


Рисунок 2.8 – Результат виконання стандартного тесту create-react-app

Тепер спробуємо додати власний тест і перевірити, чи працює автоматичний перезапуск (Рисунок 2.9).

```
it('fake test to test testing platform', () => {
  expect(true).toBe(false);
});
```

Рисунок 2.9 – Фальшивий тест

Переконуємось, що тести автоматично перезапустились (рисунок 2.10)

Рисунок 2.10 – Результат виконання доданого тесту

2.2. Налаштування ESLint

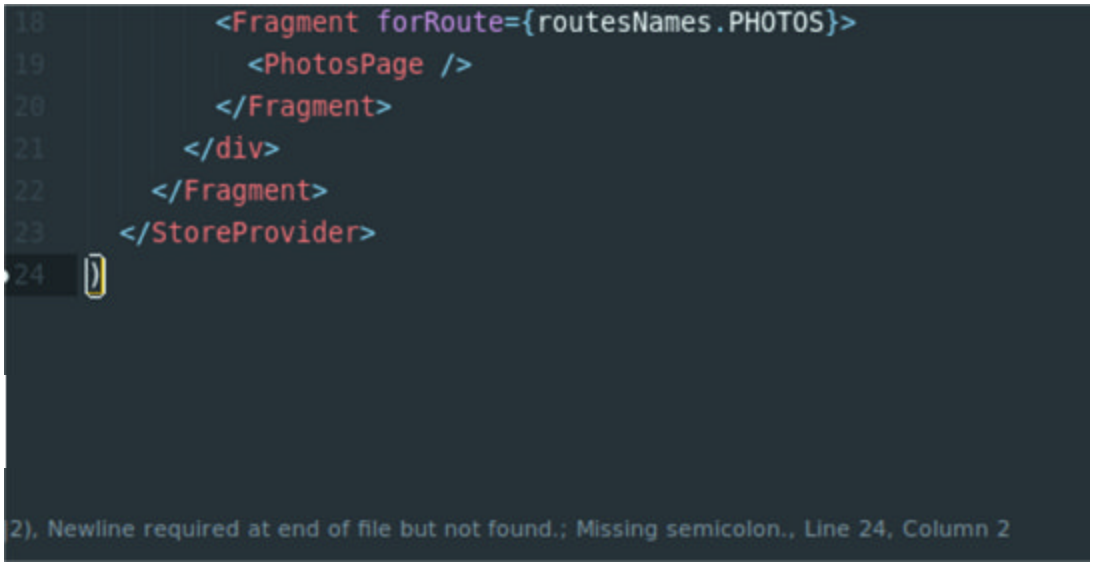
Для того, щоб швидко виявляти та виправляти невалідний код, а також щоб єдиний стиль коду залишався у всьому додатку на протязі всього часу розробки, ми використовуємо статичний аналізатор коду ESLint, який сканує наш код на наявність помилок а також на предмет того, чи він відповідає стилю, який ми задаємо в файлі налаштуваннь (.eslintrc). Налаштування ESLint для даного проекту можна побачити на рисунку 2.21.

```
module.exports = {
  env: {
    jest: true,
    browser: true,
  },
  parser: 'babel-eslint',
  plugins: ['import', 'promise', 'eslint-plugin-react', 'jest
  extends: [
    'airbnb',
    'plugin:import/errors',
    'plugin:import/warnings',
    'plugin:jest/recommended',
  ],
  globals: {
    shallow: true,
    render: true,
  },
  rules: {
    'comma-dangle': [
      'error',
      {
        arrays: 'always-multiline',
        objects: 'always-multiline',
        imports: 'always-multiline',
        exports: 'always-multiline',
        functions: 'ignore',
      },
    ],
    'max-len': [
      'error',
      {
        code: 100,
        ignoreComments: true,
        ignorePattern: '^import\\W.*',
      },
    ],
    'newline-per-chained-call': [
      'error',
      {
        ignoreChainWithDepth: 5,
      },
    ],
    'import/prefer-default-export': 0,
    'react/forbid-prop-types': 1,
    'react/no-unused-prop-types': 1,
    'react/jsx-filename-extension': 0,
    'react/display-name': 1,
    'jsx-a11y/anchor-is-valid': 0,
    'function-paren-newline': 0,
    'object-curly-newline': 0,
    'no-confusing-arrow': 0,
    'no-underscore-dangle': 0
  },
};
```

```
Column 8                    Spaces: 2    JavaScrip
```

2.11 – Вміст файлу .eslintrc

Для того, щоб бачити помилки коду відразу в редакторі, скористаємось плагіном SublimeLinter для Sublime Text, який “на льоту” аналізуватиме наш код і підсвічуватиме помилки (рисунок 2.10).



2.10 – Приклад роботи плагіна SublimeLinter

2.3. Структура проекту

Весь сирцевий код додатку знаходить в директорії src (Додаток А).

«appState» - директорія, в якій знаходиться все що пов’язано з обслуговуванням стану додатку, тобто з Redux, а саме:

• Творці дій - чисті функції, які повертають певні дії на основі своїх аргументів. Кожна дія - це об’єкт, який обов’язково повинен містити поле type, тобто назву дії. Також він може містити якусь додаткову інформацію в інших полях, яка буде використана в редукторі. В даному проекті вся додаткова інформація міститься в полі payload. Приклад об’єкту дії можна побачити на рисунку 2.11.

• Редуктори - чисті функції, які на вхід приймають об’єкт дії та попередній стан додатку і повертають наступний стан додатку. Іншими словами, редуктори описують, яким чином змінюється стан додатку в залежності від дій. Приклад реалізації редуктора можна побачити на рисунку 2.12.

Рисунок 2.11 – Оголошення творців дій

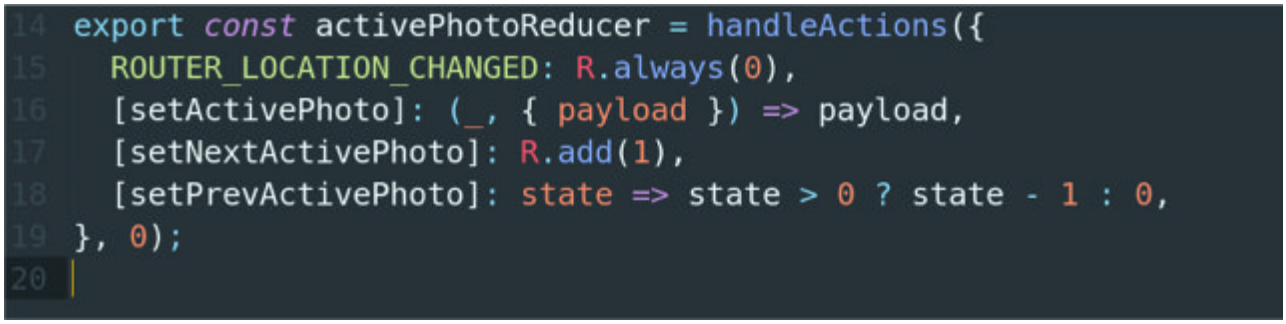


Рисунок 2.12 – Оголошення редуктора

Варто зауважити, що функції createActions та handleActions не відносяться до стандартного пакету Redux, вони взяті з пакету redux-actions. Ці функції - просто маленькі обгортки для більш зручного створення дій та редукторів, а також вони не вимагають створення окремих констант для зберігання назв дій. Без них код на рисунках вище виглядав би наступним чином (рисунок 2.13 та 2.14):

```
const SET_ACTIVE_PHOTO = 'SET_ACTIVE_PHOTO';
const SET_NEXT_ACTIVE_PHOTO = 'SET_NEXT_ACTIVE_PHOTO';
const SET_PREV_ACTIVE_PHOTO = 'SET_PREV_ACTIVE_PHOTO';

export const setActivePhoto = payload => ({
  type: SET_ACTIVE_PHOTO,
  payload,
});

export const setNextActivePhoto = () => ({
  type: SET_NEXT_ACTIVE_PHOTO,
});

export const setPrevActivePhoto = () => ({
  type: SET_PREV_ACTIVE_PHOTO,
});
```

Рисунок 2.13 – Оголошення творців дій без використання пакету redux-actions

```
const SET_ACTIVE_PHOTO = 'SET_ACTIVE_PHOTO';
const SET_NEXT_ACTIVE_PHOTO = 'SET_NEXT_ACTIVE_PHOTO';
const SET_PREV_ACTIVE_PHOTO = 'SET_PREV_ACTIVE_PHOTO';

export const activePhotoReducer = (state = 0, { type, payload }) => {
  switch(type) {
    case 'ROUTER_LOCATION_CHANGED':
      return 0;
    case SET_ACTIVE_PHOTO:
      return payload;
    case SET_NEXT_ACTIVE_PHOTO:
      return state + 1;
    case SET_PREV_ACTIVE_PHOTO:
      return state > 0 ? state - 1 : 0;
    default:
      return state;
  }
}
```

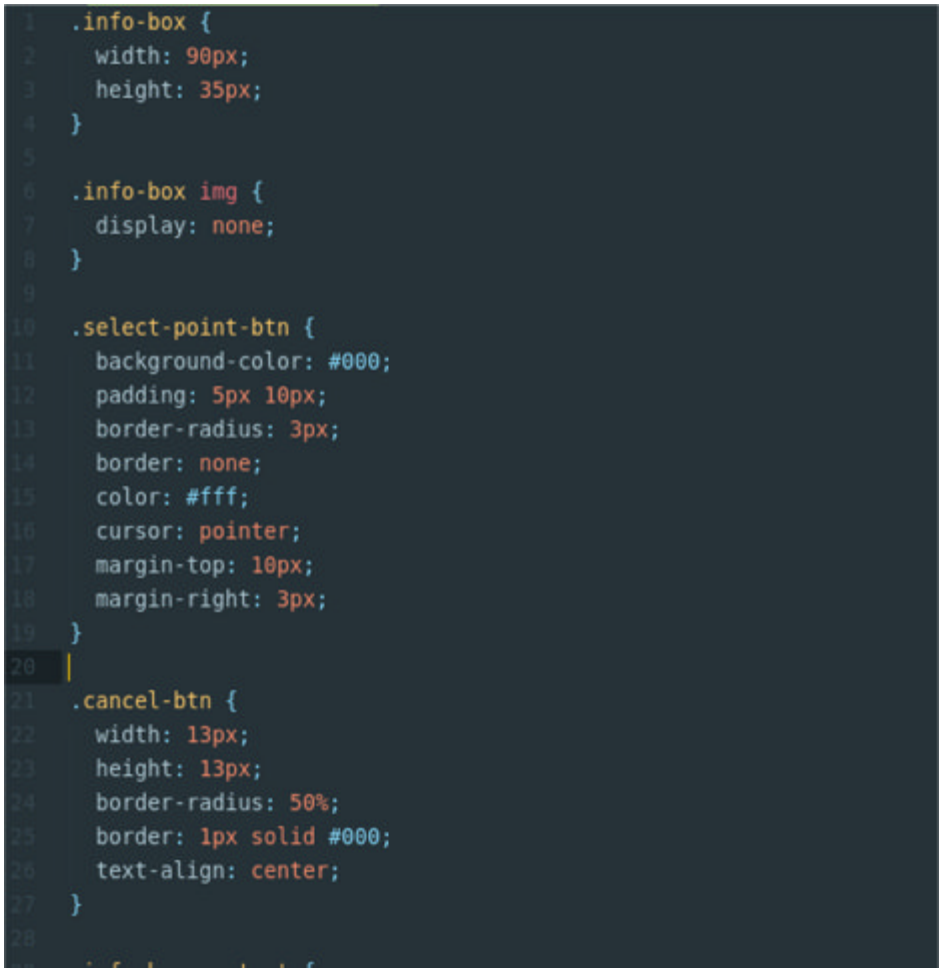
Рисунок 2.14 – Оголошення редуктора без використання пакету redux-actions

«components» - директорія, в якій зберігаються всі репрезентаційні React-компоненти додатку, разом із CSS, який динамічно імпортується. Іншими словами у цій директорії описано все, що стосується зовнішнього вигляду додатку. Компоненти не містять будь якої складної логіки, тільки приймають параметри та певним чином відмальовують їх на сторінці. Всі компоненти - це чисті функції (у даному проекті не використовуються класи для визначення компонентів). Для того, щоб полегшити підтримку та рефакторинг стилів, CSS-файли, які лежить в директорії разом з компонентом містять стилі тільки для класів, які використовуються в відповідному компоненті. Приклад компонента та його CSS можна побачити на рисунках 2.15 та 2.16 відповідно.

```
export const ConfirmPointInfoBoxC = ({
  position,
  onConfirm,
  onCancel,
}) => (
  <InfoBox
    isHidden={!!position}
    position={position}
    options={{
      boxClass: 'info-box',
      alignBottom: true,
      enableEventPropagation: true,
    }}
  >
    <div className="info-box-content">
      <button className="select-point-btn" onClick={onConfirm}>
        Select
      </button>
      <CancelIcon onClick={onCancel} className="close-icon" />
    </div>
  </InfoBox>
);

ConfirmPointInfoBoxC.propTypes = {
  position: shape({
    lat: func.isRequired,
    lng: func.isRequired,
  }).isRequired,
  onConfirm: func.isRequired,
  onCancel: func.isRequired,
};
```

2.15 – Приклад оголошення React-компонента



```css
.info-box {
  width: 90px;
  height: 35px;
}

.info-box img {
  display: none;
}

.select-point-btn {
  background-color: #000;
  padding: 5px 10px;
  border-radius: 3px;
  border: none;
  color: #fff;
  cursor: pointer;
  margin-top: 10px;
  margin-right: 3px;
}

.cancel-btn {
  width: 13px;
  height: 13px;
  border-radius: 50%;
  border: 1px solid #000;
  text-align: center;
}
```

2.16 – Приклад CSS

«containers» - директорія, в якій зберігаються контейнери для компонентів. Контейнер - це набір компонентів вищого порядку (Higher Order Components), які складені в єдиний компонент за допомогою стандартної функції compose, адже попри те, що компоненти вищого порядку - це React-компоненти, вони водночас є звичайними функціями, до яких ми можемо застосовувати паттерни функціонального програмування і робити багато цікавих речe. Контейнери іноді називають “розумними компонентами”, адже вони містять всю логіку, яка пов’язана з їхніми репрезентаційним компонентами. Оскільки в даному проекті використовується Redux, то більшість контейнерів слугує для того, щоб діставати дані з Redux-сховища, обробляти їх певним чином, та передавати компонентам. Також в контейнерах викликаються різні дії, які потрапляють в редуктор, який робить певні зміни в стані додатку. Щоб змінений стан відобразився відповідним чином на сторінці, як не дивно, нічого не потрібно

робити, адже компонент вищого порядку connect, за допомогою якого ми дістаємо дані зі сховща, є реактивним і відразу передасть нові дані репрезентаційному компоненту, як тільки вони помінялись. Здебільшого стурктура цієї директорії копіює структуру директорії components. Приклад оголошення контейнера можна побачити на рисунку 2.17.

```
export const ConfirmPointInfoBox = compose(
  connect(
    R.prop('ui'),
    { push, setInfoBoxCoords },
  ),
  renderNothingIf(
    R.complement(R.path(['infoBoxCoords', 'lat'])),
  ),
  mapProps(
    ({ infoBoxCoords, ...actions }) => ({
      position: infoBoxCoords,
      onCancel: () => actions.setInfoBoxCoords(null),
      onConfirm: () =>
        actions.push({
          pathname: routesNames.PHOTOS,
          query: fromLatLng(infoBoxCoords),
        }),
    })
  )
)(ConfirmPointInfoBoxC);
```

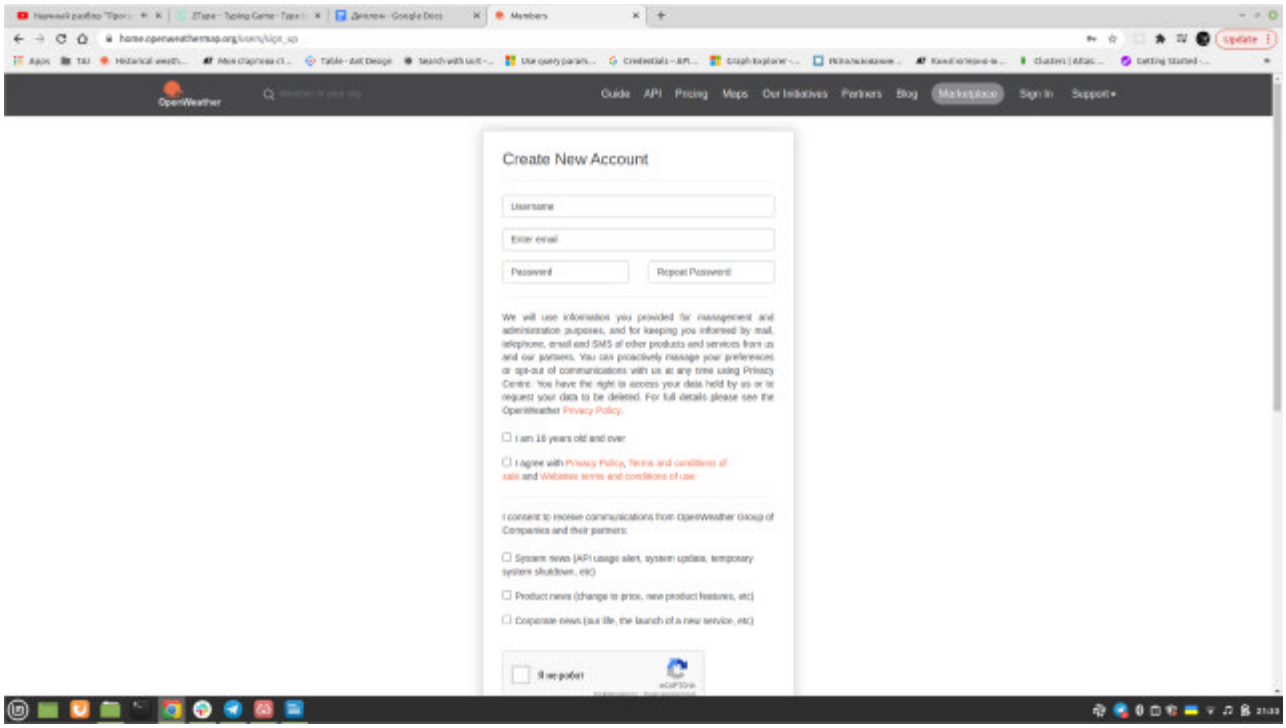
Рисунок 2.17 – Приклад оголошення контейнера

«utils» - директорія, в якій зберігаються корисні функції для форматування дат, роботи з NASA API, роботи з координатами, компоненти вищого порядку, а також різні константи.

2.4. Ключ для Open Weather

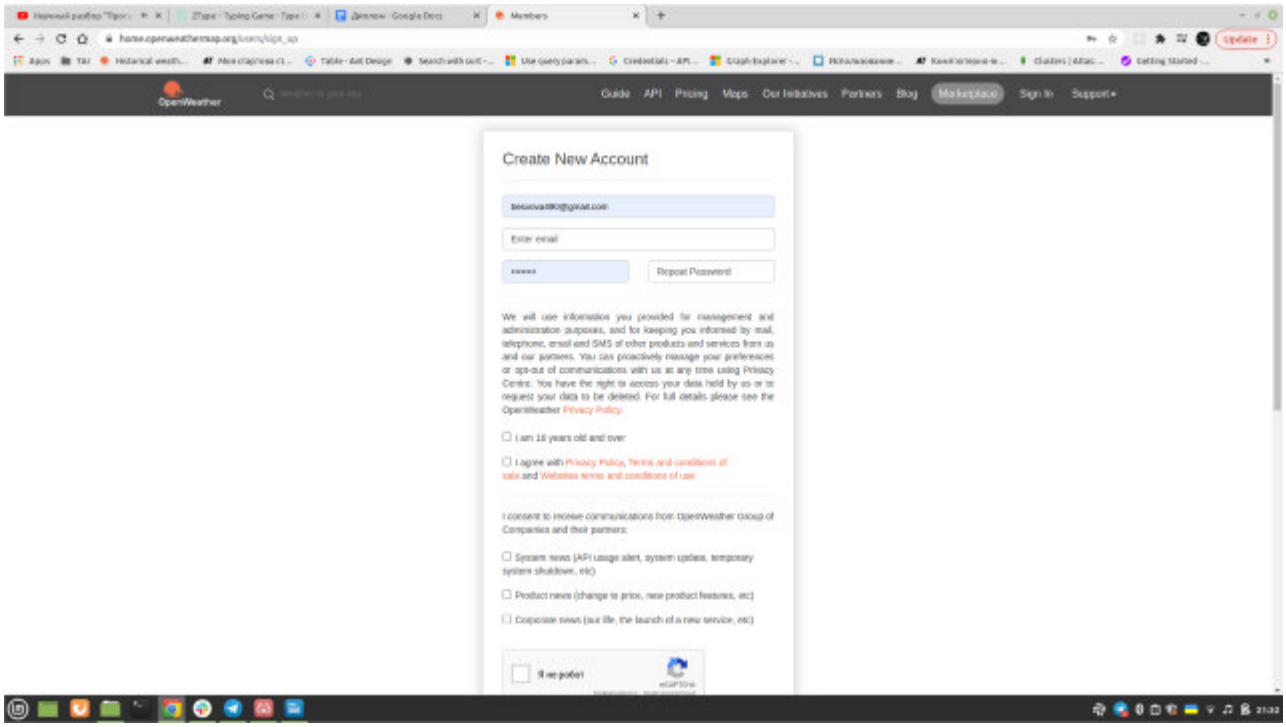
Будь яке Open Weather можна використовувати з демо-ключем, але він має досить великі обмеження у вигляді 60 запитів на хвилину та 1000000 запитів на місяцб (на одну IP-адресу), тому варто дістати власний ключ. Для того, щоб його дістати потрібно заповнити форму (рисунок 2.18), яку можна знайти за посиланням

https://openweathermap.org/price



2.19 – Форма реєстрації розробника, який хоче використовувати Open Weather

Після того, як ми заповнили дані (рисунок 2.20), повинні отримати ключ, який вставлятимемо в URL при кожному запиті. Даний ключ має обмеження в 60 запитів за годину, але цього достатньо для наших потреб.



2.20 – Заповнена форма реєстрації розробника

2.5. Ключ для Google Maps API

Для того, щоб скористатися Google Maps в нашому додатку, також потрібно зареєструватись. Для цього потрібно перейти на головну сторінку Google Maps APIs (https://cloud.google.com/maps-platform/?hl=en) та натиснути кнопку “Get Started”, після чого заповнити декілька форм і отримати ключ.

2.6. Зберігання та поширення API ключів

Для того, щоб не вставляти ключі прямо в коді, їх потрібно зберігати в окремих файлах, з яких будуть читатись і вставлятись змінні під час компіляції. Це робиться через те, що пряме використання ключів в коді є небезпечним, адже хтось інший може їх побачити та скористатись, оскільки вони будуть збережені в Git-історії. Такі змінні називаються змінними серидовища. create-react-app пропонує зберігати їх у .env файлах, чим ми і користуємось (рисунок 2.21).

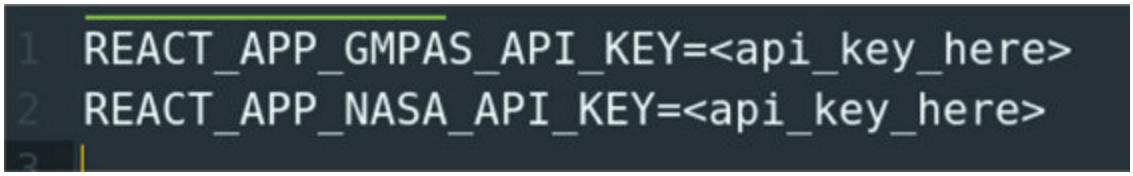


Рисунок 2.22 - Вміст файлу .env.development

Приклад доступу до цих змінних в коді показано на рисунку 2.23.

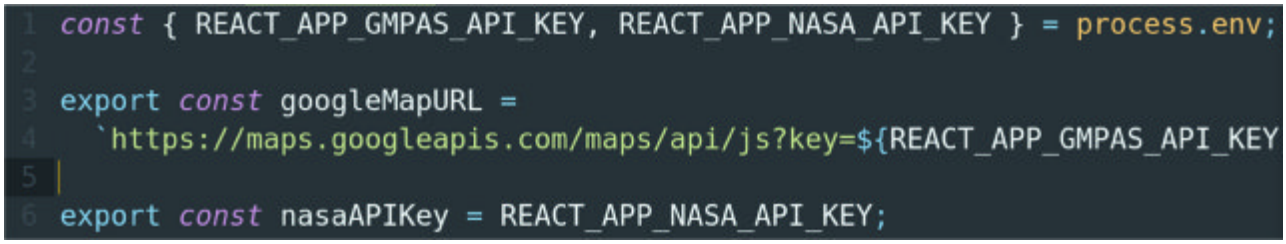


Рисунок 2.23 - Приклад використання змінних серидовища, які вказані в .env файлі

Також .env файли потрібно не забути вказати в .gitignore файлі, щоб вони не потрапляли в Git-історію. Для того, щоб передати .env файли іншому розробнику, який також хоче долучитись до розробки, можна скористатись додатком Keybase, за допомогою якого можна легко створити зашифрований Git-репозиторій і надавати доступ до нього тільки певним людям.

3 СПЕЦІАЛЬНА ЧАСТИНА

3.1. Реалізація вхідної точки додатку

Вхідна точка додатку, в якій ренедриться все React-дерево знаходиться в файлі index.js в директорії src. Вміст файлу показаний на рисунку 3.1.

```
1  import React from 'react';
2  import ReactDOM from 'react-dom';
3  import { App } from './App';
4
5  ReactDOM.render(<App />, document.getElementById('root'));
6
```

Рисунок 3.1 – Реалізація вхідної точки додатку (файл index.js в директорії src)

3.2. Роутинг та провайдери

Для реалізації роутингу в даному додатку було використано пакет redux-little-router, який побудований на новому History API і зберігає поточну URL адресу та всі параметри в Redux-сховищі, що дозволяє працювати з URL за допомогою звичайних Redux-дій, а не імперативно звертатись до API JavaScript.

Для того, щоб кожен компонент додатку міг діставати дані з Redux-сховща, потрібно огорнути весь додаток в провайдер, який за допомогою React context API передасть наше Redux-сховщие всім дочірнім компонентам.
Реалізацію роутингу та Redux-провайдера можна побачити на рисунку 3.2.

```
import React from 'react';
import { Fragment } from 'redux-little-router';
import { Provider as StoreProvider } from 'react-redux';

import { MapPage, PhotosPage } from './containers/pages';
import { routesNames } from './utils/constants';
import { store } from './appState';

import './globalStyles.css';

export const App = () => (
  <StoreProvider store={store}>
    <Fragment forRoute="/">
      <div>
        <Fragment forRoute={routesNames.MAP}>
          <MapPage />
        </Fragment>
        <Fragment forRoute={routesNames.PHOTOS}>
          <PhotosPage />
        </Fragment>
      </div>
    </Fragment>
  </StoreProvider>
);
```

Рисунок 3.2 – Реалізація роутингу та Redux-провайдера

3.3. Реалізація сторінки з картою

Реалізацію сторінки з картою можна побачити на рисунку 3.3.

Як бачимо, тут просто використовується компонент GoogleMap із пакету react-google-maps, який дає зручну React-обгортку над Google Maps API. Цей компонент приймає всього декілька властивостей, зокерма:

- defaultZoom - початковий зум карти
- defaultCenter - початкові координати центрування карти
- containerElement - елемент, в який буде огорнуто карту та mapElement
- mapElement - елемент, в який буде огорнуто карту

• onClick - функція, яка викликається при кліку на будь яку точку карти. В даному випадку в Redux-сховище записуються координати точки, на яку було клікнуто.

```
import React from 'react';
import * as R from 'ramda';
import { mapProps } from 'recompose';
import { connect } from 'react-redux';

import { GoogleMap, withGoogleMap } from 'react-google-maps';
import { setInfoBoxCoords } from '../../appState/ui';
import { ConfirmPointInfoBox } from '../ConfirmPointInfoBox';


export const MapPage = R.compose(
  connect(null, { setInfoBoxCoords }),
  mapProps(
    ({ ...actions }) => ({
      defaultZoom: 2,
      defaultCenter: { lat: 0, lng: 0 },
      containerElement: <div style={{ height: '400px' }} />,
      mapElement: <div style={{ height: '100vh' }} />,
      onClick: R.pipe(
        R.prop('latLng'),
        actions.setInfoBoxCoords,
      ),
    }),
  ),
  withGoogleMap,
)(props => (
  <GoogleMap {...props}>
    <ConfirmPointInfoBox />
  </GoogleMap>
));
```

Рисунок 3.3 – Реалізація сторінки з картою

Дочірнім елементом «GoogleMap» є «ConfirmPointInfoBox», який з’являється у відповідних координатах, які записані в Redux-сховищі. Якщо координат немає, він не показується. Реалізацію компонента та контейнера можна побачити на рисунках 3.5 та 3.6 відповідно.

```jsx
import React from 'react';
import { func, shape } from 'prop-types';
import { InfoBox } from 'react-google-maps/lib/components/addons/InfoBox';

import { CancelIcon } from '../lib/Icons';

import './styles.css';

export const ConfirmPointInfoBoxC = ({
  position,
  onConfirm,
  onCancel,
}) => (
  <InfoBox
    isHidden={!!position}
    position={position}
    options={{
      boxClass: 'info-box',
      alignBottom: true,
      enableEventPropagation: true,
    }}
  >
    <div className="info-box-content">
      <button className="select-point-btn" onClick={onConfirm}>
        Select
      </button>
      <CancelIcon onClick={onCancel} className="close-icon" />
    </div>
  </InfoBox>
);
```

Рисунок 3.5 – Реалізація компонента ConfirmPointInfoBoxC

З рисунка видно, що в реалізації ConfirmPointInfoBoxC використовується компонент InfoBox із бібліотеки react-google-maps. Цей кмпонент сулугує обгорткою над нативним Google Maps API.

```
import * as R from 'ramda';
import { compose, mapProps } from 'recompose';
import { push } from 'redux-little-router';
import { connect } from 'react-redux';
import { setInfoBoxCoords } from '../appState/ui';


import { ConfirmPointInfoBoxC } from '../components/ConfirmPointInfoBox';
import { renderNothingIf } from '../utils/hocs';
import { routesNames } from '../utils/constants';
import { fromLatLng } from '../utils/coords';


export const ConfirmPointInfoBox = compose(
  connect(
    R.prop('ui'),
    { push, setInfoBoxCoords },
  ),
  renderNothingIf(
    R.complement(R.path(['infoBoxCoords', 'lat'])),
  ),
  mapProps(
    ({ infoBoxCoords, ...actions }) => ({
      position: infoBoxCoords,
      onCancel: () => actions.setInfoBoxCoords(null),
      onConfirm: () =>
        actions.push({
          pathname: routesNames.PHOTOS,
          query: fromLatLng(infoBoxCoords),
        }),
    })
  )
)(ConfirmPointInfoBoxC);
```

3.6 – Реалізація контейнера ConfirmPointInfoBox

Як бачимо, при кліку на кнопку «Select», ми переходимо на іншу URL-адресу, вказуючи відповідні координати в параметрах. В принципі зберігати координати в адресі не обов’язково, але для того, щоб користувач міг поділитись посиланням на фотографії по певних координтах, йому достатньо скопіювати посилання, в іншому випадку прийшлося б казати, яку точку він вибирав і інший користувач повинен вибрати саме цю точку, що дуже незручно.

3.4. Реалізація сторінки з фотографіями

Реалізацію сторінки можна побачити на рисунку 3.7.

```
import React from 'react';

import { CurrentPhoto } from '../../CurrentPhoto';
import { withFetchAvailableAssets } from './hocs';
import { AllPhotos } from '../../AllPhotos';
import { BackBtn } from '../../../components/BackBtn';

import './styles.css';


export const PhotosPage = withFetchAvailableAssets(
  () => (
    <div className="photos-page-container">
      <div className="photos-page-main-section">
        <BackBtn />
        <CurrentPhoto />
        <div />
      </div>
      <div className="all-photos-container">
        <AllPhotos />
      </div>
    </div>
  )
);
```

Рисунок 3.7 – Реалізація сторінки з фотографіями

«withFetchAvailableAssets» - це компонент вищого порядку, який при componentDidMount (життєвий метод компонента) робить запит до NASA API, дістаючи дати та id всіх доступних фото для поточних координат (але не самі фото) та записує їх в Redux-сховище. Його реалізація можна глянути на рисунку 3.8.

```
import * as R from 'ramda';
import { connect } from 'react-redux';

import { setInitialAssets } from '../../../appState/photos';
import { callOn } from '../../../utils/hocs';
import { fetchAvailableAssets } from '../../../utils/nasaApi';

export const withFetchAvailableAssets = R.compose(
  connect(
    R.pipe(
      R.path(['router', 'query']),
      R.objOf('targetPointCoords'),
    ),
    { setInitialAssets },
  ),
  callOn(
    'componentDidMount',
    ({ targetPointCoords: { lat, lng }, ...actions }) =>
      fetchAvailableAssets({ lat, lon: lng })
        .then(
          R.pipe(
            R.prop('results'),
            R.map(
              R.evolve({ date: date => new Date(date) }),
            ),
            R.sort(R.descend(R.prop('date'))),
            actions.setInitialAssets,
          )
        )
  )
);
```

Рисунок 3.8 – Реалізація компонента вищого порядку withFetchAvailableAssets

Як бачимо, дочірніми компонентами PhotosPage є:

• BackBtn - кнопка повернення на сторінку карти (реалізація на рисунку 3.9)

• CurrentPhoto - компонент, який відображає поточне фото, яке переглядається (реалізація компонента на контейнера на рисунках 3.10 та 3.11 відповідно)

• AllPhotos - компонент, який відображає всі доступні фотографії (реалізація компонента на контейнера на рисунках 3.12 та 3.13 відповідно)

```
import React from 'react';
import { Link } from 'redux-little-router';

import { LeftArrowIcon } from '../lib/Icons';
import { routesNames } from '../../utils/constants';
import './styles.css';


export const BackBtn = () => (
  <Link className="icon-btn back-button" href={routesNames.MAP}>
    <LeftArrowIcon />
  </Link>
);
```

Рисунок 3.9 - Реалізація компонента BackBtn

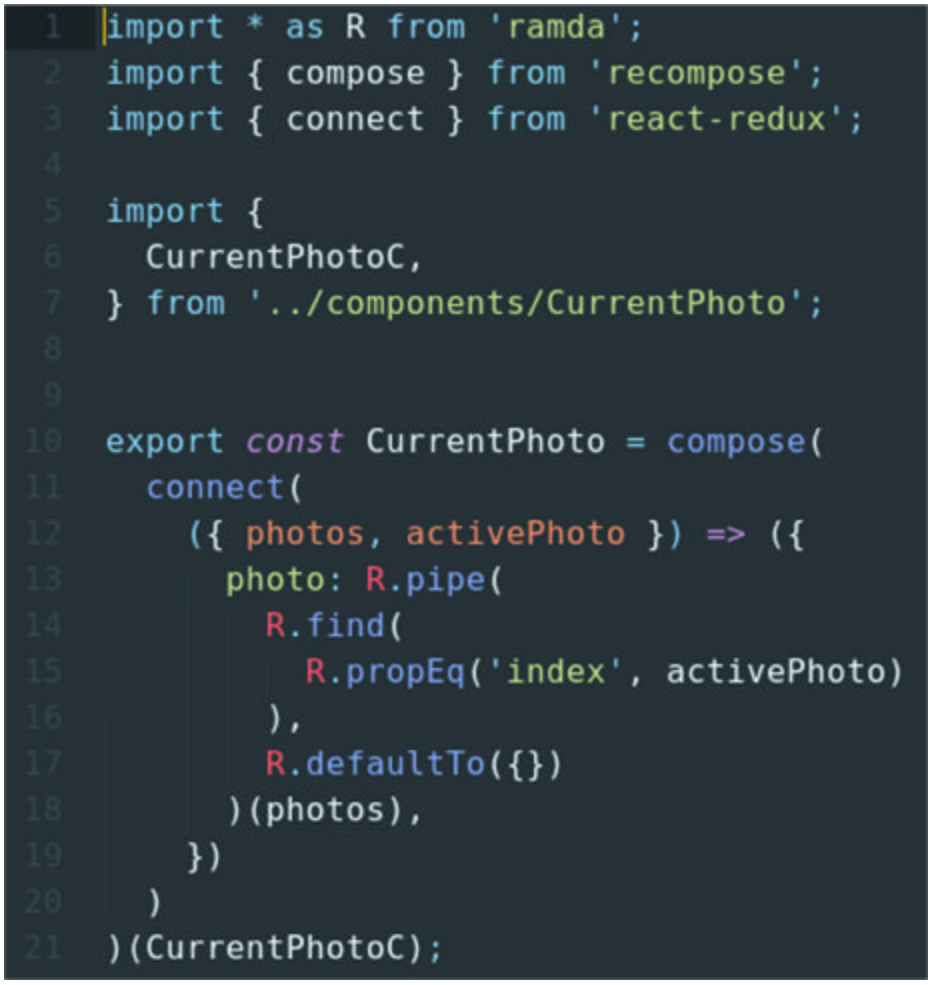
```
import React from 'react';
import { shape, string, instanceOf } from 'prop-types';

import { formatDate } from '../../utils/formatters';
import './styles.css';

export const CurrentPhotoC = ({ photo }) => (
  photo.url ?
    <div>
      <img src={photo.url} alt="pop" />
      <div className="photo-date">
        {formatDate(photo.date)}
      </div>
    </div> : null
);

CurrentPhotoC.propTypes = {
  photo: shape({
    url: string.isRequired,
    date: instanceOf(Date).isRequired,
  }).isRequired,
};
```

Рисунок 3.10 - Реалізація компонента CurrentPhotoC



```
import * as R from 'ramda';
import { compose } from 'recompose';
import { connect } from 'react-redux';

import {
  CurrentPhotoC,
} from '../components/CurrentPhoto';


export const CurrentPhoto = compose(
  connect(
    ({ photos, activePhoto }) => ({
      photo: R.pipe(
        R.find(
          R.propEq('index', activePhoto)
        ),
        R.defaultTo({})
      )(photos),
    })
  )
)(CurrentPhotoC);
```

Рисунок 3.11 – Реалізація контейнера CurrentPhoto

```
import React, { Fragment } from 'react';
import { number, arrayOf, object } from 'prop-types';

import { PhotoItem } from '../../containers/AllPhotos/PhotoItem';
import { MorePhotosBtn } from '../../containers/MorePhotosBtn';

import './styles.css';


export const AllPhotosC = ({ photos, activePhoto }) => (
  <Fragment>
    {
      photos.map((photo, index) => (
        <PhotoItem
          key={photo.id}
          index={index}
          photo={photo}
          isActive={photo.index === activePhoto}
        />
      )
    )}
    <MorePhotosBtn />
  </Fragment>
);

AllPhotosC.propTypes = {
  photos: arrayOf(object),
  activePhoto: number.isRequired,
};

AllPhotosC.defaultProps = {
  photos: [],
};
```

Рисунок 3.12 – Реалізація компонента AllPhotosC

```
import * as R from 'ramda';
import { connect } from 'react-redux';

import { setPrevActivePhoto, setNextActivePhoto } from '../../appState/activePhoto';

import { AllPhotosC } from '../../components/AllPhotos';
import { assocWith } from '../../utils/general';
import { withArrowClicksListeners } from './hocs';


export const AllPhotos = R.compose(
  connect(
    R.pipe(
      R.pick([
        'activePhoto',
        'photos',
        'ui',
      ]),
      assocWith(
        'photos',
        R.converge(R.take, [
          R.pathOr(0, ['ui', 'visiblePhotosCount']),
          R.propOr([], 'photos'),
        ])
      ),
    ),
    {
      setPrevActivePhoto,
      setNextActivePhoto,
    }
  ),
  withArrowClicksListeners,
)(AllPhotosC);
```

Рисунок 3.13 – Реалізація контейнера AllPhotos

4 ОХОРОНА ПРАЦІ

4.1 Аналіз потенційних шкідливих впливів на працівників, що працюють з ЕОМ

Аналіз виробничого травматизму показує, що кількість травм, які спричинені дією електричного струму є незначною і складає близько 1 %, однак із загальної кількості смертельних нещасних випадків частка електротравм вже складає 20-40 % і займає одне з перших місць. Найбільша кількість випадків електротравматизму, в тому числі із смертельними наслідками, стається при експлуатації електроустановок напругою до 1000 В, що пов'язано з їх поширенням і відносною доступністю практично для кожного, хто працює на виробництві.

Основними причинами електротравматизму на виробництві є: випадкове доторкання до неізольованих струмопровідних частин електроустаткування; використання несправних ручних електроінструментів; робота без надійних захисних засобів та запобіжних пристосувань; доторкання до незаземлених корпусів електроустаткування, що опинилися під напругою внаслідок пошкодження ізоляції; недотримання правил улаштування, технічної експлуатації та правил техніки безпеки при експлуатації електроустановок, ЕОМ та інших електричних пристроїв [9].

Електроустаткування, з яким доводиться мати справу практично всім працівникам на виробництві, становить значну потенційну небезпеку ще й тому, що органи чуття людини не здатні на відстані виявляти наявність електричної напруги. В зв'язку з цим захисна реакція організму проявляється лише після того, як людина потрапила під дію електричної напруги. Проходячи через організм людини електричний струм справляє на нього термічну, електролітичну, механічну та біологічну дію.

В таблиці 4.1 наведено порогові значення змінного та постійного струму, що дуже важливо знати працівникам, які безпосередньо взаємодіють з електронною апаратурою чи мають до неї доступ.

Таблиця 4.1 - Порогові значення змінного та постійного струму

Вид струму	Пороговий відчутний	Пороговий невідпускаючий струм, мА	Пороговий фібриляційний струм, мА
Змінний струм частотою 50 *Гц*	0,5-1,5	6-10	80-100
Постійний струм	5,7-7,0	50-80	300

Шкіливий вплив на організм працівників, що працюють з ЕОМ також чинять різного роду електромагнітні випромінювання. Їх продукують монітори з електронно-променевою трубкою і в значно меншій мірі самі схеми пристрою. Наприклад в таблиці 4.2 подані види випромінювання електронно-променевих трубок і відповідні нормовані значення [10].

Види випромінювань	**Діапазон**	**Фактичні (середні) дані замірів**	**Нормовані значення**
Рентгенівське	понад 1,2 КеВ	9-10-12 мкр/г	75,0 мкр/г
Ультрафіолетове випромінювання	220-280 нм 280-320 нм	0 мкр/г 0-0,02 мкр/г	0,01 Вт/м 0,01 Вт/м
Видимий діапазон	320-400 нм 400-700 нм	0,1-2,0 мкр/г 2,5-4,0 мкр/г	10 Вт/м
ІЧ-випромінювання	700 нм-1 мм	0,05-4,0 мкр/г	100 Вт/м
Електростатичне поле	0 Гц	15 кВ/м	20-60 кВ/м
Електричний струм	50 Гц	U=220В I=2А	U=220В I=0,1 А

Таблиця 4.2 - Види випромінювання електронно-променевих трубок

Електромагнітне випромінювання може викликати біологічні та функціональні несприятливі ефекти в організмі людини. Функціональні ефекти проявляються у передчасній втомлюваності, частих болях голови, погіршенні сну, порушеннях центральної нервової та серцево-судинної систем. При систематичному опроміненні ЕМП спостерігаються зміни кров'яного тиску,

сповільнення пульсу, нервово-психічні захворювання, деякі трофічні явища (випадання волосся, ламкість нігтів та ін.). Сучасні дослідження вказують на те, що радіочастотне випромінювання, впливаючи на центральну нервову систему, є вагомим стрес-чинником [9].

Біологічні несприятливі ефекти впливу електромагнітного випромінювання проявляються у тепловій та нетепловій дії. Нині достатньо вивченою можна вважати лише теплову дію ЕМП, яка призводить до підвищення температури тіла та місцевого вибіркового нагрівання органів та тканин організму внаслідок переходу електромагнітної енергії у теплову. Таке нагрівання особливо небезпечне для органів із слабкою терморегуляцією (головний мозок, око, нирки, шлунок, кишківник, сім'яники). Наприклад, випромінювання сантиметрового діапазону призводять до появи катаракти, тобто до поступової втрати зору.

Механізм та особливості нетеплової дії ЕМП радіочастотного діапазону ще до кінця не з'ясовані. Частково таку дію пояснюють специфічним впливом радіочастотного випромінювання на деякі біофізичні явища: біоелектричну активність, що може призвести до порушення усталеного протікання хімічних та ферментативних реакцій, вібрацію субмікроскопічних структур, енергетичне збудження (часто резонансне) на молекулярному рівні, особливо на конкретних частотах.

Не слід забувати і про шум. Особливу увагу привертають ЕОМ старіших модифікацій, де спостерігається доволі високий показник шумності. Це пов’язано з використанням великої кількості охолоджуючих вентиляторів (кулерів), а також і з недосконалістю самих електричних вузлів ЕОМ [10].

Так за даними медиків дія шуму може спричинити нервові, серцево-судинні захворювання, виразкову хворобу, порушення обмінних процесів та функціонування органів слуху тощо.

Важливим фактором є освітленість. Якщо вона недостатня чи навпаки спостерігається надлишок – то це може призвести до великих проблем із зором.

Адже відомо, що майже, 90% всієї інформації про довкілля людина одержує через органи зору. Під час здійснення будь-якої трудової діяльності втомлюваність очей, в основному, залежить від напруженості процесів, що супроводжують зорове сприйняття. При поганому освітленні людина швидко втомлюється, працює менш продуктивно, зростає потенційна небезпека помилкових дій і нещасних випадків. Основними причинами поганої освітленості: погане планування розміщення вікон, погане планування штучного освітлення, використання неякісних моніторів, що зумовлюється не виконанням норм [9].

4.2 Основні вимоги з охорони праці до користувачів ЕОМ та їх робочого місця

При експлуатації ПК необхідно пам'ятати, що первинні мережі електроспоживання під час роботи знаходяться під напругою, яка є небезпечною для життя людини, тому необхідно користуватися справними розетками, відгалужувальними та з'єднувальними коробками, вимикачами та іншими електроприладами. До роботи з ПК допускаються працівники, з якими проведений вступний інструктаж та первинний інструктаж (на робочому місці) з питань охорони праці, техніки безпеки, пожежної безпеки та зроблений запис про їх проведення у спеціальному журналі інструктажів. Працівники при роботі з ПК повинні дотримуватися вимог техніки безпеки, пожежної безпеки. При виявленні в обладнанні ПК ознак несправності (іскріння, пробоїв, підвищення температури, запаху гару, ознак горіння) необхідно негайно припинити роботи, відключити усе обладнання від електромережі і терміново повідомити про це відповідних посадових осіб, спеціалістів. Потрібно знати місця розташування первинних засобів пожежегасіння, план евакуації працівників, матеріальних цінностей з приміщення в разі виникнення пожежі.

Перед початком роботи на ПК користувач повинен:

- пересвідчитися у цілості корпусів і блоків (обладнання) ПК;

• перевірити наявність заземлення, справність і цілість кабелів живлення, місця їх підключення.

Забороняється вмикати ПК та починати роботу при виявлених несправностях.

Під час роботи, пересвідчившись у справності обладнання, увімкнути електроживлення ПК, розпочати роботу, дотримуючись умов інструкції з її експлуатації [10].

Великий вплив на умови праці здійснює приміщення, в якому безпосередньо працюють люди з ЕОМ. До таких приміщень є ряд вимог:

• стіни приміщень для роботи з ПК мають бути пофарбовані чи обклеєні шпалерами пастельних кольорів з коефіцієнтом відбиття 40 - 60 %. У випадках, коли такі приміщення зорієнтовані на південь, вікна повинні обладнуватися сонцезахисними пристроями (жалюзі, штори і т. п.);

• для освітлення приміщень з ПК необхідно використовувати люмінесцентні світильники. Освітленість робочих місць у горизонтальній площині на висоті 0,8 м від підлоги повинна бути не менше 400 лк. Вертикальна освітленість у площині екрану не більше 300 лк;

• у приміщеннях для роботи з ПК необхідно проводити щоденне вологе прибирання та регулярне провітрювання протягом робочого дня.

Також висуваються окремі вимоги до робочого місця:

• робочі місця для працюючих з дисплеями необхідно розташовувати таким чином, щоб до поля зору працюючого не потрапляли вікна та освітлювальні прилади. Відео термінали повинні встановлюватися під кутом 90 - 105 градусів до вікон та на відстані, не меншій 2,5 - 3 м від стіни з вікнами;

• до поля зору працюючого з дисплеєм не повинні потрапляти поверхні, які мають властивість віддзеркалювання. Покриття столів повинне бути матовим з коефіцієнтом 0,25 - 0,4;

• відстань між робочими місцями з ПК повинна бути не меншою 1,5 м у ряду та не меншою 1 м між рядами. ПК повинні розміщуватися не ближче 1 м від джерела тепла;

• відстань від очей користувача до екрану повинна становити 500 - 700 мм, кут зору - 10 - 20°, але не більше 40°, кут між верхнім краєм відео терміналу та рівнем очей користувача повинен бути меншим 10°. Найбільш вигідне є розташування екрану перпендикулярно до лінії зору користувача;

• монітор обладнується захисною плівкою, що розсіює шкідливе електромагнітне випромінювання.

• оптимальні розміри робочої поверхні стільниці 1600×900 мм. Під стільницею робочого столу повинно бути вільний простір для ніг із розмірами по висоті не менше 600мм, по ширині 500мм, по глибині 650мм;

• всі ЕОМ повинні бути заземлені [9].

4.3 Безпечна експлуатація електроустановок та пожежна безпека

Робота щодо забезпечення безпечної експлуатації електроустановок здійснюється згідно з обов'язковими, для всіх споживачів електроенергії, незалежно від їх відомчої приналежності, правилами технічної експлуатації електроустановок споживачів та правилами техніки безпеки при експлуатації електроустановок споживачів. Обслуговування діючих електроустановок, проведення в них оперативних переключень, організація та виконання ремонтних, монтажних, налагоджувальних робіт і випробувань здійснюються спеціально підготовленим електротехнічним персоналом.

Безпечна експлуатація електроустановок забезпечується: конструкцією електроустановок; технічними способами та засобами захисту; організаційними та технічними заходами.

Конструкція електроустановок повинна відповідати умовам їх експлуатації та забезпечувати захист персоналу від можливого доторкання до рухомих та струмовідних частин, а устаткування - від потрапляння всередину сторонніх предметів та води.

За способом захисту людини від ураження електричним струмом встановлено п'ять класів електротехнічних виробів: 0, 01,1, II, III. До класу 0 належать вироби, які мають робочу ізоляцію і у яких відсутні елементи для

заземлення. До класу 01 належать вироби, які мають робочу ізоляцію, елемент для заземлення та провід без заземлювальної жили для приєднання до джерела живлення. До класу І належать вироби, які мають робочу ізоляцію та елемент для заземлення. Якщо виріб класу І має кабель до джерела живлення, то цей кабель повинен мати заземлювальну жилу та штепсельну вилку зі заземлювальним контактом. Цей контакт є дещо довшим за робочі контакти вилки для того, щоб забезпечувати випереджальне замикання заземлювального контакту під час увімкнення та більш запізніле розмикання його під час вимикання. До класу ІІ належать вироби, які мають подвійну чи посилену ізоляцію і не мають елементів для заземлення. До класу ІІІ належать вироби, які не мають внутрішніх та зовнішніх електричних кіл з напругою понад 42 В [10].

Технічні способи та засоби захисту (ТСЗЗ) поділяють на:

1) ТСЗЗ при нормальних режимах роботи електроустановок (ізоляція струмовідних частин, забезпечення недоступності неізольованих струмовідних частин, попереджувальні сигналізація, знаки та написи, застосування малих напруг, захисне розділення електромереж, вирівнювання потенціалів);

2) ТСЗЗ при переході напруги на металеві нормально неструмовідні частини електроустановок (захисні заземлення, занулення, вимикання);

3) електрозахисні засоби та запобіжні пристосування.

Забезпечення пожежної безпеки — це один із важливих напрямків щодо охорони життя та здоров'я людей.

Основною причиною пожеж у приміщеннях, де використовуються ЕОМ є в основному короткі замикання, які виникають внаслідок неправильного монтажу або екс-плуатації електроустановок, старіння або пошкодження ізоляції. Струм короткого замикання залежить від потужності джерела струму, відстані від джерела струму до місця замикання та виду замикання. Великі струми замикання викликають іскріння та нагрівання струмопровідних частин до високої температури, що може викликати займання ізоляції провідників та горючих будівельних конструкцій, які знаходяться поряд.

Ще однією причиною є струмові перевантаження, що виникають при ввімкненні до мережі додаткових споживачів струму або при зниженні напруги в мережі. Тривале перевантаження призводить до нагрівання провідників, що може викликати займання ізоляції [10].

Продуктами згорання стають багато речовин, високі концентрації яких можуть серйозно впливати на організм людини (таблиця 4.3)

Таблиця 4.3 – Степені небезпечності концентрацій продуктів згорання

Речовини	Концентрація					
	смертельна за умови вдихання протягом 5 - 10 хв.		небезпечна (отруйна) за умови вдихання протягом 0,5-1,0 год.		переносима за умови вдихання протягом 0,5-1,0 год.	
	%	г/м³	%	г/м³	%	г/м³
Оксид азоту	0,05	1,0	0,01	0,2	0,005	0,1
Оксид вуглецю	0,5	6,0	0,2	2,4	0,1	1,2
Вуглекислий газ	9,0	162	5,0	90	3,0	54
Сірчаний газ	0,3	8,0	0,04	ІД	0,01	0,3
Сірководень	0,08	1,1	0,04	0,6	0,02	0,3
Сірковуглець	0,2	6,0	0,1	3,0	0,05	1,5
Хлористий вуглець	0,3	4,5	0,1	1,5	0,01	0,15
Синильна кислота	0,02	0,2	0,01	0,1	0,005	0,05

Дим являє собою велику кількість видимих найдрібніших твердих та (або) рідинних часточок незгорівших речовин, що знаходяться в газах у завислому стані. Він викликає інтенсивне подразнення органів дихання та слизових оболонок (сильний кашель, сльозотечу тощо). Крім того, у задимлених приміщеннях внаслідок погіршення видимості сповільнюється евакуація людей, а часом провести її зовсім неможливо.

При пожежі електроустаткування в приміщеннях використовуються вуглекислотні вогнегасники типу ОУ-2, ОУ-5, ОУ-8 ємністю 2,5 - 8 літрів, які призначені для гасіння пожеж всіх видів. Також невелику ділянку пожежі можна локалізувати методом пониження доступу кисню в осередок вогню

накинувши на нього азбестове полотно або грубу шерстяну тканину. Перелік засобів гасіння або локалізації вогню наводяться в таблиці 4.4.

Таблиця 4.4 – Засоби гасіння вогню

Назва приміщення	Площа, яка захищається, м2	Типи первинних засобів пожежогасіння	Кількість, шт.
Комп'ютерний зал	100	Вуглекислотні вогнегасники типу ОУ-8	2
		Азбестове полотно (або кошма) 1×1, 2×1 або 2×2 м	4

Пожежний захист і вибухозахист забезпечуються правильним вибором ступеня вогнестійкості окремих елементів і конструкцій; обмеженням розповсюдження вогню у випадку виникнення пожежі; впровадженням систем активного поглинання вибуху; застосування систем протидимового захисту; забезпеченням безпечної евакуації людей; застосуванням засобів пожежної сигналізації, оповіщення і пожежогасіння; організацією пожежної охорони об'єкта.

Обов'язковим є інструктаж персоналу з пожежної безпеки [9].

4.4 Характеристика та розрахунок захисного заземлення електроустановок

Захисне заземлення – допоміжне електричне з'єднання з землею чи її еквівалентом металевих неструмоведучих частин, що можуть виявитися під напругою.

Мета захисного заземлення – знизити напругу дотику між корпусом електроустановки і землею до 42В, і менше що там виникає в результаті ушкодження чи пробою ізоляції струмоведучих частин.

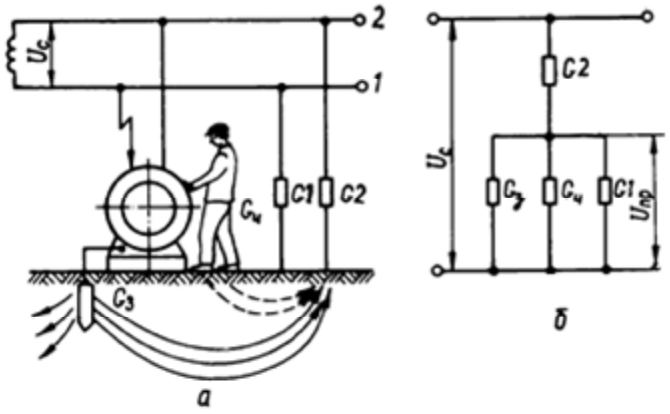
Захисне заземлення варто відокремити від робітника і заземлення для захисту від розрядів статичної та атмосферної електрики.

Робоче заземлення – допоміжне з'єднання з землею нейтральних точок обмоток генераторів, силових і вимірювальних трансформаторів, дугогасних апаратів та інших ланцюгів з метою забезпечення нормальної роботи

електроустановок. Заземлення для захисту від розрядів статичної й атмосферної електрики здійснюється для відводу цих зарядів у землю [10].

Розглянемо принцип роботи захисного заземлення. На рисунку 4.1,а показано ситуація дотик людини до заземленого корпуса електроустановки, на якому з’явилася напруга, а на рисунку 4.1,б – її еквівалентна електрична схема.

Спочатку визначимо значення напруги дотику Uдот, що прикладається до людини при дотику її до заземленого корпуса, з одного боку, і до ніг, з іншого, а потім значення струму I, що протікає через людину в цьому ланцюзі.



а – схема дотику; б – еквівалентна електрична схема заземленої електроустановки

Рисунок 4.1 – Дотик людини до заземленого корпуса електроустановки:

Для спрощення математичних перетворень оперувати будемо провідностями, а потім їх замінимо опорами.

Провідність заземлення Gз, провідність людини Gл і провідність ізоляції G1 проводу 1 щодо землі включені паралельно між собою і послідовно з провідністю G2 ізоляції проводу 2.

Сумарна провідність паралельного ланцюга складе:

$G_{лан} = G_з + G_л + G_1$.

Провідність усього ланцюга

$$G=\frac{G_{лан}G_2}{G_{лан}+G_2}=\frac{(G_з+G_л+G_1)G_2}{G_з+G_л+G_1+G_2}$$

Напруга Uдот, що впливає на людину при дотику до корпусу електроустановки

$$\frac{U_{дот}}{U_м}=\frac{G}{G_{лан}}=\frac{(G_з+G_л+G_1)G_2}{(G_з+G_л+G_1+G_2)(G_л+G_1+G_з)}=\frac{G_2}{G_з+G_л+G_1+G_2}$$

де Uм – напруга мережі, В.

Тоді

$$U_{дот}=\frac{U_м G_2}{(G_з+G_л+G_1+G_2)}$$

Провідності Gл, G1, G2 набагато менше провідності заземлення Gз і ними як доданками в знаменнику можна знехтувати. Замінюючи провідності опорами і приймаючи r2 = rіз (rіз – опір ізоляції), одержимо

$$U_{дот}=\frac{U_м r_з}{r_{із}}=I_з r_з \qquad (4.1)$$

де Iз = Uм/rіз.

Аналіз виразу (4.1) дозволяє стверджувати, що найбільш доступним заходом щодо зниження напруги Uдот є зменшення опору заземлення rз, а збільшувати опір ізоляції економічно недоцільно.

Струм, що протікає через людину при дотику її до заземленого корпуса електроустановки [10]:

$$I_л=\frac{U_{дот}}{R_л}=\frac{U_м r_з}{R_л r_{із}} \qquad (4.2)$$

Виконуємо розрахунок захисного заземлення електроустановки.

Визначаємо кількість заземлювачів для заземлюючого пристрою підстанції. Зі сторони U1=11 кВ нейтраль ізольована, струм замикання на землю Iз=12А. На стороні U2=0,6 кВ нейтраль глухо заземлена. Опір природних заземлювачів Rпр=13Ом. Питомий опір ґрунту ρвим=0,9·104.

Заземлювачі вертикальні стрижневі діаметром 10мм довжиною 5м, коефіцієнт підвищення опору ґрунту ψ=2.

1. Визначаємо необхідний нормативний опір заземлюючого пристрою:

• для мережі 11 кВ,

$$R_з = \frac{U_з}{I_з} = \frac{125}{12} = 10,4 Ом$$

• для мережі 0,6 кВ, згідно ПУЕ

$R_з \leq 4$ Ом .

Приймаємо менше значення опору Кз=4 Ом, оскільки заземлюючий пристрій спільний, як для мережі 11 кВ так і для 0,6 кВ.

2. Визначаємо опір штучних заземлювачів:

$$R_{шт} = \frac{R_з R_{пр}}{R_{пр} - R_з} = \frac{4 \cdot 13}{13 - 4} = 5,78$$ Ом.

В якості штучних заземлювачів приймаємо стрижневі електроди діаметром 10 мм і довжиною 5 м.

Визначаємо опір розтіканню струму одного заземлювача:

R0(рс)=0,00227·ρвим·ψ = 0,00227·0,9·104·2 = 40,86Ом.

3. Визначаємо кількість заземлювачів:

$$R_{шт} = \frac{R_з R_{пр}}{R_{пр} - R_з} = \frac{4 \cdot 13}{13 - 4} = 5,78$$ шт.

де Кb=0,56 – коефіцієнт використання електродів.

Приймаємо n=13 електродів.

4. Перевіряємо загальний опір заземлення:

$$R_{шт} = \frac{R_з R_{пр}}{R_{пр} - R_з} = \frac{4 \cdot 13}{13 - 4} = 5,78$$ Ом;

$$R_з = \frac{R_{пр} \cdot R_{шт}}{R_{пр} + R_{шт}} = \frac{13 \cdot 5{,}61}{13 + 5{,}61} = 3{,}92$$ Ом.

Значить кількість електродів вибрана вірно

ВИСНОВКИ

Під час роботи над дипломним проектом було проведено розробку веб-додатку для генерації графіків погоди. Для розробки використовувались популярні веб-технології та бібліотеки, такі як React та Redux. Код написаний у функціональному стилі за допомогою бібліотеки Ramda.

Актуальність реалізації даної системи полягає в необхідності можливості зручно та швидко будувати графіки а також в збереженні цих даних з метою проведення презентації презентації, лабораторних робіт або ж власного використання та дослідження.

Значними перевагами розробленої веб-системи перед іншими, являються її повна безкоштовність, інтеграції з соціальними мережами та зручний та інтуїтивно зрозумілий користувацький інтерфейс.

Внаслідок проведеної роботи було виконано всі поставленні завдання та вимоги. Розроблено зручний інтерфейс користувача, що підтверджено консультаціями з досвідченими галузевими дизайнерами. Програмне забезпечення розроблене в процесі виконання дипломної роботи відповідає сучасним тенденціям технологічного розвитку.

Виходячи з розрахунків та тестування функціоналу додатку можна зробити висновок що додаток вийшов зручним та швидким.

Функціональні можливості додатку дозволяють користувачам створювати свої графіки погоди чи користуватись існуючими з додатку та створювати власні шаблони які користувач може доповнювати сам.

Отже, завдання дипломного проектування виконано і результати виконання можна впроваджувати в сферу інформаційно-комунікаційних технологій.

ПЕРЕЛІК ПОСИЛАНЬ

1. Orientation Modeling of Bernoulli Gripper Device with Off-Centered Masses of the Manipulating Object / V. Savkiv, R. Mykhailyshyn, O. Fendo, M. Mykhailyshyn // Procedia Engineering. – 2017. – № 187. – P. 264 – 271. – DOI: 10.1016/j.proeng.2017.04.374.
2. Justification of Design and Parameters of Bernoulli-Vacuum Gripping Device / V. Savkiv, R. Mykhailyshyn, F. Duchon, O. Fendo // International Journal of Advanced Robotic Systems. – 2017. – DOI: 1729881417741740.
3. Energy efficiency analysis of the manipulation process by the industrial objects with the use of Bernoulli gripping devices / V. Savkiv, R. Mykhailyshyn, F. Duchon, M. Mikhalishin // Journal of Electrical Engineering. – 2017. – № 68 (6). – P. 496 – 502. – DOI: 10.1515/jee-2017-0087.
4. Experimental Research of the Manipulatiom Process by the Objects Using Bernoulli Gripping Devices / R. Mykhailyshyn, V. Savkiv, M. Mikhalishin, F. Duchon // In Young Scientists Forum on Applied Physics and Engineering, International IEEE Conference. – 2017. – P. 8 – 11. – DOI: 10.1109/YSF.2017.8126583.
5. Modeling of Bernoulli gripping device orientation when manipulating objects along the arc / V. Savkiv, R. Mykhailyshyn, F. Duchon, M. Mikhalishin // International Journal of Advanced Robotic Systems. – 2018. – DOI: 1729881418762670.
6. Investigation of the energy consumption on performance of handling operations taking into account parameters of the grasping system / R. Mykhailyshyn, V. Savkiv, F. Duchon, V. Koloskov, I. Diahovchenko // 2018 IEEE 3rd International Conference on Intelligent Energy and Power Systems (IEPS) – IEEE, 2018. – P. 295 – 300. – DOI: 10.1109/ieps.2018.8559586.
7. Analysis of frontal resistance force influence during manipulation of dimensional objects / R. Mykhailyshyn, V. Savkiv, F. Duchon, V. Koloskov, I.

Diahovchenko // 2018 IEEE 3rd International Conference on Intelligent Energy and Power Systems (IEPS) – IEEE, 2018. – P. 301 – 305. – DOI: 10.1109/ieps.2018.8559527.

8. Substantiation of Bernoulli Grippers Parameters at Non-Contact Transportation of Objects with a Displaced Center of Mass / R. Mykhailyshyn, V. Savkiv, F. Duchon, P. Maruschak, O. Prentkovskis // 22nd International Scientific Conference Transport Means 2018. – Klaipeda, 2018. – P. 1370 – 1375.
9. Gasdynamic analysis of the Bernoulli grippers interaction with the surface of flat objects with displacement of the center of mass / V. Savkiv, R. Mykhailyshyn, F. Duchon // Vacuum. – 2019. – № 159, P. 524 – 533. – DOI: 10.1016/j.vacuum.2018.11.005.
10. Protection of Digital Power Meters Under the Influence of Strong Magnetic Fields / R. Mykhailyshyn, V. Savkiv, I. Diahovchenko, R. Olsen, D. Danylchenko // 2019 IEEE 2nd Ukraine Conference on Electrical and Computer Engineering UKRCON-2019 – IEEE, 2019. – P. 314 – 320. – DOI: 10.1109/UKRCON.2019.8879985.
11. Research of Energy Efficiency of Manipulation of Dimensional Objects With the Use of Pneumatic Gripping Devices / R. Mykhailyshyn, V. Savkiv, I. Diahovchenko, F. Duchon, R. Trembach // 2019 IEEE 2nd Ukraine Conference on Electrical and Computer Engineering UKRCON-2019 – IEEE, 2019. – P. 527 – 532. – DOI: 10.1109/UKRCON.2019.8879957.
12. Diahovchenko, I., Lebedynskyi, I., Mykhailyshyn, R., & Savkiv, V. (2019, September). Methods to Improve the Accuracy of Power Meters through the Application of Nanomaterials and Calibration Techniques. In 2019 IEEE 9th International Conference Nanomaterials: Applications & Properties (NAP) (pp. 02NEE17-1). IEEE. doi: 10.1109/NAP47236.2019.216994.
13. Rogowsky coil applications for power measurement under non-sinusoidal field conditions / I. Diahovchenko, R. Mykhailyshyn, D. Danylchenko, S. Shevchenko // Energetika. – 2019. – 65(1), P. 14 – 20. – DOI: 10.6001/energetika.v65i1.3972.

14. Control of a small quadrotor for swarm operation / A. Trizuljak, F. Duchoň, J. Rodina, A. Babinec, M. Dekan, R. Mykhailyshyn // Journal of Electrical Engineering. – 70(1). – 2019. – P. 3-15. – DOI: 10.2478/jee-2019-0001.
15. Optimization of design parameters of Bernoulli gripper with an annular nozzle / V. Savkiv, R. Mykhailyshyn, P. Maruschak, L. Chovanec, E. Prada, I. Virgala, O. Prentkovskis // Transport Means - Proceedings of the International Conference. – 2019. – P. 423-428.
16. The analysis of influence of a nozzle form of the Bernoulli gripping devices on its energy efficiency / V. Savkiv, R. Mykhailyshyn, P. Maruschak, F. Duchon, L. Chovanec // Proceedings of ICCPT 2019, May 28-29, 2019. – Tern. : TNTU, Scientific Publishing House “SciView”, 2019. – P. 66–74. – DOI: 10.5281/zenodo.3387275.
17. Usage of Light-Emitting-Diode Lamps in Decorative Lighting / R. Mykhailyshyn, I. Belyakova, V. Medvid, V. Piscio, O. Shkodzinsky, M. Markovych // IEEE 20th International Conference on Computational Problems of Electrical Engineering (CPEE). – IEEE, 2019. – DOI: 10.1109/CPEE47179.2019.8949154.
18. Justification of Influence of the Form of Nozzle and Active Surface of Bernoulli Gripping Devices on Its Operational Characteristics / V. Savkiv, R. Mykhailyshyn, P. Maruschak, F. Duchon // TRANSBALTICA XI: Transportation Science and Technology. Lecture Notes in Intelligent Transportation and Infrastructure. – Springer, 2020. — P. 263–272. – DOI: 10.1007/978-3-030-38666-5_28.
19. Analysis of Operational Characteristics of Pneumatic Device of Industrial Robot for Gripping and Control of Parameters of Objects of Manipulation / V. Savkiv, R. Mykhailyshyn, P. Maruschak, F. Duchon, O. Prentkovskis, I. Diahovchenko // TRANSBALTICA XI: Transportation Science and Technology. Lecture Notes in Intelligent Transportation and Infrastructure. – Springer, 2020. — P. 504–510. – DOI: 10.1007/978-3-030-38666-5_53.

20. Progress and Challenges in Smart Grids: Distributed Generation, Smart Metering, Energy Storage and Smart Loads / Diahovchenko, I., Kolcun, M., Čonka, Z., Savkiv, V., Mykhailyshyn, R. // Iranian Journal of Science and Technology, Transactions of Electrical Engineering, – 2020. – P. 1-15.
21. Михайлишин Р.І. Обґрунтування параметрів та орієнтації струминного захоплювача маніпулятора для автоматизації вантажно-розвантажувальних операцій: автореф. дис. на здобуття наук. ступеня канд. техн. наук : спец. 05.05.05 “Піднімально-транспортні машини” / Р.І. Михайлишин. – Тернопіль, 2018. – 21 с.
22. Михайлишин Р. І. Optimization of bernoulli gripping device’s orientation under the process of manipulations along direct trajectory / Р.І. Михайлишин, Я. І. Проць, В.Б. Савків // Вісник ТНТУ. – Тернопіль, 2016. – Том 81. – № 1. – С. 107 – 117.
23. Михайлишин Р. І. Аналіз методів планування траєкторій маніпуляторів / Р.І. Михайлишин, В.Б. Савків // Збірник наукових праць «Перспективні технології та прилади» Луцький НТУ. – Луцьк, 2016. – №8 (1). – С. 61 – 69.
24. Justification of the object of manipulation parameters influence on the optimal orientation and lifting characteristics of Bernoulli gripping device / В.Б. Савків, Р.І. Михайлишин, Ф. Духон, М.С. Михайлишин // Вісник Херсонського національного технічного університету. – Херсон, 2017. – № 2 (61). – С. 98 – 104.
25. «Ознайомлення з основними функціями програмного середовища RobotStudio» : методичні вказівки до лабораторної роботи № 1 з курсу “Гнучкі комп'ютеризовані системи та робототехніка” для студентів спеціальності 151 «Автоматизація та комп’ютерно-інтегровані технології» / укл. Р.І. Михайлишин, В.Б. Савків. – Тернопіль : ТНТУ імені Івана Пулюя, 2019. – 45 с.
26. «Визначення базових точок та траєкторії промислового робота» : методичні вказівки до лабораторної роботи № 2 з курсу “Гнучкі

комп'ютеризовані системи та робототехніка” для студентів спеціальності 151 «Автоматизація та комп’ютерно-інтегровані технології» / укл. : Р. І. Михайлишин, В. Б. Савків. – Тернопіль : ТНТУ імені Івана Пулюя, 2019. – 17 с.

27. Методичні вказівки до лабораторної роботи № 3 «Імпорт тривимірних моделей та створення захоплювального пристрою в програмному середовищі RobotStudio» з курсу “Гнучкі комп'ютеризовані системи та робототехніка” для студентів спеціальності 151 «Автоматизація та комп’ютерно-інтегровані технології» / укл. : Р. І. Михайлишин, В. Б. Савків. – Тернопіль : ТНТУ імені Івана Пулюя, 2019. – 24 с.
28. «Робота з віртуальним пультом управління FlexPendant в програмному середовищі RobotStudio» методичні вказівки до лабораторної роботи № 4 з курсу “Гнучкі комп'ютеризовані системи та робототехніка” для студентів спеціальності 151 «Автоматизація та комп’ютерно-інтегровані технології» / укл. : Р. І. Михайлишин, В. Б. Савків. – Тернопіль : ТНТУ імені Івана Пулюя, 2019. – 23 с.
29. «Операції над об’єктами та контроль зіткнень в програмному середовищі RobotStudio» методичні вказівки до лабораторної роботи № 5 з курсу “Гнучкі комп'ютеризовані системи та робототехніка” для студентів спеціальності 151 «Автоматизація та комп’ютерно-інтегровані технології» / укл. : Р.І. Михайлишин, В.Б. Савків. – Тернопіль: ТНТУ імені Івана Пулюя, 2019. – 34 с.
30. «Розробка механізму конвеєра та програмування операцій MultiMove в програмному середовищі RobotStudio» методичні вказівки до лабораторної роботи № 6 з курсу “Гнучкі комп'ютеризовані системи та робототехніка” для студентів спеціальності 151 «Автоматизація та комп’ютерно-інтегровані технології» / укл. : Р. І. Михайлишин, В. Б. Савків. – Тернопіль : ТНТУ імені Івана Пулюя, 2019. – 39 с.
31. «Створення роботизованої станції в програмному середовищі RobotStudio» методичні вказівки до лабораторної роботи № 7 з курсу

“Гнучкі комп'ютеризовані системи та робототехніка” для студентів спеціальності 151 «Автоматизація та комп’ютерно-інтегровані технології» / укл. : Р. І. Михайлишин, В. Б. Савків. – Тернопіль : ТНТУ імені Івана Пулюя, 2019. – 19 с.

32. «Розробка роботизованої лінії для автоматизації вантажно-розвантажувальних операцій в програмному середовищі RobotStudio» методичні вказівки до лабораторної роботи № 8 з курсу “Гнучкі комп'ютеризовані системи та робототехніка” для студентів спеціальності 151 «Автоматизація та комп’ютерно-інтегровані технології» / укл. : Р. І. Михайлишин, В. Б. Савків. – Тернопіль : ТНТУ імені Івана Пулюя, 2019. – 24 с.