КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп'ютерних наук та кібернетики
Кафедра інтелектуальних програмних систем

Кваліфікаційна робота
на здобуття освітнього рівня бакалавра
за спеціальністю 121 Інженерія програмного забезпечення
на тему:

РОЗРОБКА РЕАКТИВНОЇ ФРОНТЕНД БІБЛІОТЕКИ З ЕЛЕМЕНТАМИ ТРАНСПІЛЯЦІЇ КОДУ

Виконав студент 4-го курсу
Андрій БАРТІШ _____
(підпис)

Науковий керівник:
асистент, кандидат фіз.-мат. наук
Костянтин ЖЕРЕБ _____
(підпис)

Засвідчую, що в цій роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент _____
(підпис)

Роботу розглянуто й допущено до захисту на засіданні кафедри інтелектуальних програмних систем
« 29 » травня 2023 р.,
протокол № 11
Завідувач кафедри
Олександр ПРОВОТАР _____
(підпис)

Київ – 2023

РЕФЕРАТ

Обсяг роботи 48 сторінок, 8 ілюстрацій, 15 джерел посилань.

TYPESCRIPT, WEBPACK, ОПТИМІЗАЦІЯ, РЕАКТИВНЕ ПРОГРАМУВАННЯ, ТРАНСПІЛЯЦІЯ КОДУ, ФРОНТЕНД БІБЛІОТЕКА.

Об’єктом дослідження є сучасні методи розробки веб-інтерфейсів та їх основні проблеми. Предметом роботи є система, для реактивного оновлення HTML(HyperText Markup Language).

Метою роботи є створення власної фронтенд бібліотеки, з використанням транспіляції коду та реактивним оновленням HTML компонентів. Програми написані з використанням даної бібліотеки повинні займати менший обсяг дискового простору та пам’яті процесора в порівнянні з ідентичними застосунками написаними з використанням найпопулярнішої бібліотеки для створення фронтенд-додатків.

Інструменти розроблення: редактор коду Visual Studio Code, платформа запуску JavaScript-коду Node.js, менеджер пакетів npm, веб-браузер Google Chrome.

Результати роботи: налаштовано збирач(module bundler) для створення єдиного мініфікованого вихідного пакету програми та транспілятор для роботи із JSX та TypeScript-кодом. Створено бібліотеку, що реактивно оновлює HTML компоненти. Використовуючи дану бібліотеку та найпопулярніший аналог, створено два ідентичні сайти та оцінено ефективність роботи системи, представлений в кваліфікаційній роботі.

Створена програма допомагає писати більш ефективні веб-інтерфейси, що займають менше дискового простору. Подальший розвиток та дослідження полягають в збільшенні та покращенні існуючих утиліт а також додавання SSR.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

AJAX — Asynchronous JavaScript And XML, підхід до побудови користувацьких інтерфейсів веб застосунків, за яких веб сторінка не перезавантажується;

DOM — Document Object Model, об'єктна модель документа;

HTML — HyperText Markup Language, мова розмітки;

LTS — Long term support, довготривала підтримка версій;

SEO — Search engine optimization, пошукова оптимізація сайту;

SPA — Single-page application, односторінковий застосунок, також відомий як односторінковий інтерфейс.

ВСТУП

Оцінка стану об’єкта розробки. Сучасний фронтенд пройшов довгий шлях він статичних сторінок до масштабних Single Page Application. Спочатку користувачі отримували розмітку у відповіді від сервера, потім розробники стали використовувати Client-Side Rendering та генерувати HTML через JavaScript код. Наступною ланкою розвитку сучасного вебу було повернення до коренів і використання Server-Side Rendering для оптимізації SEO(Search engine optimization) та метаданих. Тепер сервер виконував JavaScript-код, генеруючи розмітку, яка повертатиметься браузеру. Зрештою, все прийшло до змішаного використання: частина сторінки генерується на клієнті, інша частина — на сервері для збільшення швидкості та SEO, а після гідрується уже у веб-браузері.

Це все було б неможливим без створення та використання сучасних фронтенд бібліотек та фреймворків. Хоча їх існує досить багато і всі вони непогано виконують своє завдання, все ж потреби користувачів зростають і технологіям, які існують досить давно, важко з ними справлятися. Підтримка попередніх версій бібліотек, а також принципи закладені в архітектуру коду не дають з усією потужністю виконувати специфічні задачі таким рішенням.

Актуальність роботи. Рішення для створення фронтенду, що існують сьогодні — це досить великі інструменти, які для своєї роботи використовують значну частину обсягу кінцевого фронтенд-застосунка, його процесорного часу та пам’яті. Крім того їм важко поєднувати зрозумілий синтаксис чи архітектуру побудови програми та її швидкість виконання. Це все робить такі засоби непридатними для невеликих застосунків, які потребують високої швидкості виконання програми.

Мета й завдання роботи. Метою роботи є створення власної фронтенд бібліотеки, з використанням транспіляції коду та реактивним оновленням HTML компонентів. Програми написані з використанням даної бібліотеки повинні займати менший обсяг дискового простору та пам’яті процесора в порівнянні з ідентичними застосунками написаними з використанням найпопулярнішої бібліотеки для створення фронтенд-додатків. Робота потребує виконання наступних задач:

— Налаштування збирача для створення єдиного мінімального пакету застосунка;

— Використання утиліти babel для транспіляції TypeScript та JSX коду;

— Створення реактивної моделі оновлення HTML компонентів;

— Додавання утиліт для роутингу та менеджингу станом;

— Написання однакового сайту з використанням даної бібліотеки та найпопулярнішого аналога — React.js;

— Порівняння швидкості та розміру пакетів обох застосунків.

Об'єкт, методи й засоби розроблення. Об’єктом роботи є бібліотека для створення фронтенд частини веб-застосунків.

При створенні бібліотеки було використано мови програмування JavaScript та TypeScript, платформу виконання коду Node.js, менеджер пакетів npm, утиліту nvm, пакувальник WebPack, бібліотеку для транспіляції коду babel, бібліотеку реактивного програмування RxJS.

Можливі сфери застосування. Бібліотека ідеально підходить для створення невеликих застосунків, яким критично важливий мінімальний розмір кінцевого пакету і, водночас, швидкість написання веб-застосунків.

1 ЯК ВЛАШТОВАНИЙ СУЧАСНИЙ ФРОНТЕНД

1.1 Single Page Application

Single Page Application — це веб-додаток, який працює у браузері та взаємодіє з користувачем, перезавантажуючи лише частину сторінки, замість повного перезавантаження[1]. SPA(Single-page application) надає зручну та багатофункціональну користувацьку інтерактивність, працюючи в межах однієї HTML сторінки. Основні особливості SPA:

— односторінкова архітектура: SPA складається з однієї HTML сторінки, на якій відбувається динамічна зміна вмісту без перезавантаження сторінки;

— AJAX(Asynchronous JavaScript And XML) і асинхронне завантаження даних: Використовуючи AJAX, SPA взаємодіє з сервером, завантажує дані та оновлює вміст сторінки без перезавантаження;

— роутинг: SPA використовує механізм роутингу для переходу між різними сторінками та відображенням відповідного вмісту;

— використання JavaScript-фреймворків або бібліотек: Багато SPA будуються з використанням JavaScript-фреймворків або бібліотек, таких як Angular, React або Vue.js. Ці інструменти надають потужні можливості для організації структури додатку та керування станом;

Переваги SPA:

— швидкість: SPA відповідає швидким відгукам користувача, оскільки вони завантажують лише необхідні дані та оновлюють частину сторінки;

— зручність та багатофункціональність: SPA дозволяє створювати багатофункціональні додатки зі зручним та інтуїтивно зрозумілим інтерфейсом;

— менша витрата трафіку: SPA зменшує витрату трафіку, оскільки вона завантажує лише необхідні дані, а не всю сторінку заново;

— покращена користувацька взаємодія: SPA забезпечує плавну інтерактивність без помітних затримок при взаємодії з користувачем, що покращує загальний досвід використання додатка;

— багатофункціональність: SPA дозволяє реалізувати складну функціональність, таку як реактивне оновлення стану, асинхронну валідацію форм, реал-тайм оновлення даних та багато іншого.

Недоліки SPA:

— довший час очікування першого завантаження: SPA потребує завантаження всього коду додатку при першому відкритті сторінки, що може збільшити час завантаження;

— SEO-оптимізація: Оскільки вміст сторінки генерується динамічно за допомогою JavaScript, це може створювати проблеми з індексацією пошуковими системами. Однак існують підходи до розв'язання цієї проблеми, наприклад, застосування пререндерингу або серверного рендерингу;

— обмеження на стороні клієнта: SPA виконується в середовищі браузера, що може мати обмеження на ресурси, такі як оперативна пам'ять і обробник. Великі та складні SPA можуть споживати значну кількість ресурсів та впливати на продуктивність пристроїв з обмеженими можливостями.

Загальний аналіз вимог та проектування SPA передбачає врахування цих факторів та вибір оптимальних рішень для досягнення бажаних цілей щодо функціональності та продуктивності

1.2 Огляд сучасних фронтенд бібліотек та фреймворків

На сьогоднішній день існує безліч сучасних фронтенд фреймворків та бібліотек, які допомагають розробляти потужні та ефективні веб-додатки. Ось кілька з них:

— React — найпопулярніша бібліотека, яка дозволяє будувати користувацькі інтерфейси за допомогою компонентної архітектури. Він пропонує використання віртуального DOM(Document Object Model) для швидкого оновлення інтерфейсу та забезпечує велику екосистему додаткових бібліотек та інструментів[2];

— Angular — це повноцінний фреймворк, який надає широкий функціонал для розробки веб-додатків. Він має вбудовану систему модулів, роутинг, обробку форм, а також можливості для роботи з HTTP-запитами та багато іншого[3];

— Vue.js — це прогресивний фреймворк, який поєднує простоту використання з потужними можливостями. Він надає можливість створювати компоненти, використовувати директиви, робити реактивні оновлення та має зручний синтаксис;

— Ember.js — це фреймворк, який зосереджується на конвенціях та надає готові рішення для багатьох типових задач, таких як маршрутизація, керування станом та робота з шаблонами.

1.3 Огляд React.js

React є найпопулярнішою бібліотекою для розробки фронтенд-застосунків. Він набув великої популярності завдяки своїй ефективності, гнучкості та широким можливостям. Основні причини популярності React включають:

— компонентна архітектура: React базується на компонентній моделі, що дозволяє розбити інтерфейс на невеликі, повторно використовувані компоненти. Це спрощує розробку, тестування та підтримку коду;

— віртуальний DOM: React використовує віртуальний DOM для ефективного оновлення інтерфейсу. Він здатний розпізнавати та оновлювати лише ті частини сторінки, які змінилися, що поліпшує продуктивність додатків[4];

— широка спільнота та екосистема: React має велику та активну спільноту розробників, яка створює багато корисних бібліотек, інструментів та плагінів для спрощення розробки. Також, існує багато матеріалів, документації та онлайн-курсів, які допомагають вивчити React;

— підтримка з боку Facebook: React розробляється та підтримується компанією Facebook, що гарантує активний розвиток, оновлення та вдосконалення бібліотеки;

— можливість використання на стороні сервера: React може використовуватись для серверного рендерингу, що дозволяє отримати більш швидку першу загрузку сторінки та покращити індексацію пошуковими системами.

1.4 Концепція віртуального DOM та його проблеми

Віртуальний DOM — це концепція, яка використовується в бібліотеці React для оптимізації процесу оновлення веб-сторінок. Вона базується на ідеї створення віртуального представлення структури DOM у пам'яті, яке використовується для порівняння та ефективного оновлення реального DOM. Схема роботи продемонстрована на рисунку 1.1.

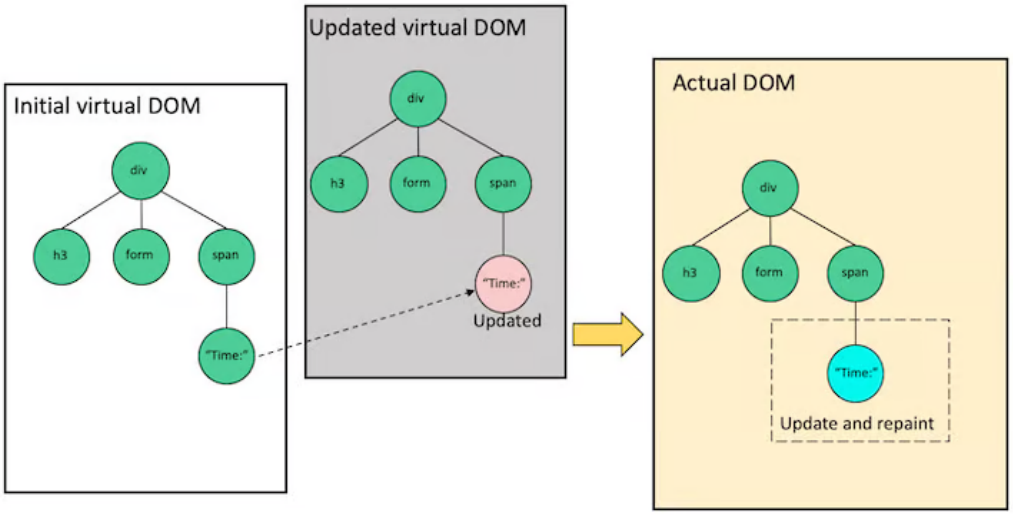

Рисунок 1.1 — Принцип роботи віртуального DOM[4]

Основна ідея полягає в тому, що React створює віртуальне дерево компонентів, яке представляє структуру реального DOM. Кожен раз, коли стан компонента змінюється, React порівнює віртуальний DOM зі старим віртуальним DOM і виявляє зміни, які потрібно зробити в реальному DOM. Це дає можливість оновлювати лише ті частини сторінки, де відбулися зміни, замість повного оновлення всього документа[5]. Проте, використання віртуальний DOM також має деякі проблеми:

— витратність ресурсів: Порівняння віртуального DOM з реальним DOM та оновлення сторінки може бути часом та ресурсомоємним процесом, особливо в разі великих та складних дерев компонентів;

— затримки в оновленні: В деяких випадках, коли віртуальне дерево стає складним та глибоким, може виникати затримка в рендерингу та оновленні сторінки. Це особливо помітно при великій кількості компонентів або при великому потоці даних;

— додатковий об'єм коду: Використання віртуального DOM вимагає включення додаткового коду, включаючи бібліотеку React та код для

обробки та оновлення віртуального дерева. Це може збільшити розмір пакету та час завантаження сторінки;

— комплексність вирішення проблеми "критичного пульсу" (Critical Render Path): Віртуальний DOM може створювати складнощі з оптимізацією критичного шляху рендерингу. При складних деревах компонентів і великій кількості залежностей можуть виникати зайві рендери, що призводить до зайвих обчислень та затримок відображення змін на сторінці;

— обмеження при роботі зі сторонніми бібліотеками: Використання віртуального DOM може вимагати модифікації сторонніх бібліотек, які використовують прямий доступ до реального DOM. Це може створювати проблеми при інтеграції з такими бібліотеками, а також ускладнювати процес міграції на React;

— потреба в додаткових знаннях та навичках: Віртуальний DOM вимагає розуміння його концепцій та механізмів роботи, а також навичок оптимізації та управління рендерингом. Це може ставити виклики перед розробниками, особливо початківцями, які ще не мають достатнього досвіду з фреймворком React.

2 ОГЛЯД ОБРАНИХ ТЕХНОЛОГІЙ

1.1 Платформа Node.js

Окрім коду бібліотеки, який повинен виконуватись на стороні клієнта, система оперує кодом, що транспілює та створює пакет для проекта. Для збереження консистентності, а також для надання кінцевому розробнику можливість розширювати функціонал, були обрано Node.js як платформу для виконання серверного коду.

Node.js — це вільне та відкрите середовище виконання JavaScript, побудоване на рушії V8 JavaScript Engine, розробленому компанією Google. Воно дозволяє виконувати JavaScript-код на сервері, що раніше було характерно тільки для виконання на браузері[6]. Node.js надає зручність розробки серверних додатків засобами JavaScript, дозволяючи розширювати функціональність серверів та реалізовувати мережеві додатки. Основні особливості Node.js:

— неблокуюча модель виконання: Node.js використовує неблокуючу систему вводу/виводу (non-blocking I/O) та подієву модель виконання, що дозволяє обробляти багато запитів одночасно без блокування потоків. Це робить його особливо ефективним для створення масштабованих мережевих додатків;

— серверні можливості: За допомогою Node.js можна легко створювати високопродуктивні сервери. Він надає вбудований модуль http, що дозволяє створювати та обробляти HTTP-запити. Крім того, за допомогою пакетного менеджера npm, доступного з Node.js, можна встановлювати та використовувати різноманітні модулі для побудови серверних додатків;

— модульна система: Node.js використовує модульну систему

CommonJS та ES модулів, що дозволяє організовувати код у модулі, що можуть бути повторно використані та імпортовані в інших частинах програми. Це сприяє покращенню організації та підтримки коду;

— велика спільнота: Node.js має велику спільноту розробників, яка активно підтримує фреймворки, бібліотеки та інструменти, що допомагають в розробці веб-додатків. Наприклад, популярні фреймворки, такі як Express.js, дозволяють швидко створювати веб-додатки з Node.js;

— незалежність від платформи: Node.js підтримується на багатьох платформах, включаючи Windows, macOS та різні дистрибутиви Linux. Це дозволяє розробникам використовувати Node.js на різних операційних системах залежно від їх потреб;

— розширені можливості: Node.js також дозволяє розширяти його функціональність за допомогою додаткових модулів. Ці модулі можуть включати різноманітні розширення для мережевого програмування, роботи з базами даних, обробки зображень та багато іншого.

Node.js використовується для розробки різних типів додатків, включаючи веб-сервери, API, розподілені системи та інші. Він здатен ефективно обробляти великі обсяги запитів та дозволяє швидко реагувати на зміни в реальному часі. Багато компаній та проектів використовують Node.js для розробки швидких та масштабованих додатків.

1.2 Менеджер пакетів npm

npm (Node Package Manager) є менеджером пакетів для JavaScript та середовища виконання Node.js. Він є стандартним інструментом для

керування залежностями, установки, оновлення та видалення пакетів у проектах, розроблених з використанням Node.js[7]. Основна функціональність npm включає наступні можливості:

— керування залежностями: npm дозволяє оголосити залежності вашого проекту у файлі package.json. Ви можете вказати пакети, які ваш проект використовує, разом із відповідними версіями. Після цього ви можете встановити всі залежності однією командою, і npm автоматично завантажить та встановить ці пакети;

— установка та оновлення пакетів: За допомогою npm ви можете легко встановлювати пакети з репозиторію npm. Просто вказуйте ім'я пакета у команді npm install, і npm завантажить та встановить його. Ви також можете оновлювати встановлені пакети до останніх версій за допомогою команди npm update;

— локальні та глобальні пакети: npm дозволяє встановлювати пакети локально для конкретного проекту або глобально для всієї системи. Локальні пакети зберігаються у директорії проекту і можуть бути використані лише в межах цього проекту. Глобальні пакети встановлюються системно і доступні з будь-якого місця на вашому комп'ютері;

— скрипти: npm дозволяє визначати та виконувати скрипти у вашому проекті. Ви можете вказати різні команди, які будуть виконуватися за допомогою команд npm run. Це дозволяє автоматизувати певні завдання, такі як збірка проекту, запуск тестів, розгортання і т. д;

д) публікація пакетів: За допомогою npm ви можете публікувати свої власні пакети в репозиторію npm. Ви можете створювати власні модулі та бібліотеки, упаковувати їх у пакети та публікувати для використання іншими розробниками;

— завантаження залежностей для розробки: npm також дозволяє вказати залежності, які використовуються лише під час розробки, а не в продакшені. Ви можете вказати такі залежності у окремому розділі devDependencies у файлі package.json.

Це лише кілька ключових можливостей, які надає npm. Він є важливим компонентом у розробці JavaScript-додатків та екосистемі Node.js, і його використовують мільйони розробників по всьому світу.

1.2 Утиліта nvm

Утиліта nvm (Node Version Manager) є інструментом, який дозволяє керувати версіями Node.js на вашому комп'ютері. Вона дозволяє встановлювати, перемикатися між різними версіями Node.js та керувати пакетами, пов'язаними з конкретною версією[8]. Основні особливості та переваги використання nvm:

— установка версій Node.js: nvm дозволяє легко встановлювати різні версії Node.js на вашому комп'ютері. Ви можете встановити більше однієї версії та використовувати їх для різних проектів. Це особливо корисно, коли потрібно розробляти або тестувати програмне забезпечення на різних версіях Node.js;

— перемикання між версіями: За допомогою nvm ви можете легко перемикатися між встановленими версіями Node.js. Це дозволяє використовувати певну версію для певного проекту без необхідності переустановки або переконфігурування системи;

— управління пакетами: nvm також дозволяє керувати пакетами, пов'язаними з конкретною версією Node.js. Ви можете легко встановлювати, оновлювати та видаляти пакети, що використовуються у вашому проекті, залежно від активної версії

Node.js;

— підтримка для різних операційних систем: nvm підтримує різні операційні системи, включаючи Windows, macOS та Linux. Це робить його універсальним інструментом, який можна використовувати на різних робочих оточеннях;

— легкість використання: nvm має простий синтаксис команд і надає зрозумілу документацію. Це робить його легким у використанні навіть для новачків.

nvm є корисним інструментом для розробників, які працюють з Node.js, особливо коли потрібно керувати різними версіями Node.js та їх пов'язаними пакетами. Він дозволяє ефективно керувати середовищем розробки та забезпечує зручну роботу з різними проектами, які використовують різні версії Node.js.

1.2 Мова програмування Typescript

TypeScript — це мова програмування, що розширює JavaScript, додаючи до нього статичну типізацію та деякі інші функції. Вона розроблена компанією Microsoft і вважається суперсетом JavaScript, що означає, що будь-який дійсний код JavaScript є також дійсним кодом TypeScript[9]. TypeScript компілюється до чистого JavaScript і може бути використаний в будь-якому середовищі, де використовується JavaScript. Основні особливості TypeScript:

— статична типізація: Одним з найбільших переваг TypeScript є можливість визначати типи для змінних, параметрів функцій, значень, що повертаються та інших елементів програми. Це дозволяє виявляти помилки на етапі компіляції, а не під час виконання програми. Статична типізація сприяє полегшенню розробки,

поліпшенню надійності коду та забезпеченню кращої підтримки редакторів коду та інструментів розробки;

— розширена підтримка ООП: TypeScript підтримує об'єктно-орієнтоване програмування (ООП) та включає функції, такі як класи, спадкування, інтерфейси, абстрактні класи та модифікатори доступу. Це дозволяє розробникам створювати більш структуровані та повторно використовувані кодові бази;

— перелічувані типи: TypeScript дозволяє визначати перелічувані типи, які встановлюють набір можливих значень для змінної. Це допомагає уникнути помилок, пов'язаних з неправильними значеннями;

— інтерфейси: TypeScript надає можливість визначати інтерфейси — контракти, які вказують наявність певних властивостей та методів у об'єктах. Це допомагає покращити контроль над структурою даних та підтримувати хорошу документацію коду;

— налаштування компіляції: TypeScript надає розширені можливості налаштування процесу компіляції. Можна визначити цільову версію JavaScript, вибрати модульну систему (таку як CommonJS або ES Modules), налаштувати показники якості коду та багато іншого;

— нтеграція з існуючим JavaScript: Оскільки TypeScript є суперсетом JavaScript, існуючий код JavaScript можна поступово впроваджувати в проект, додаючи типи поступово. TypeScript надає можливість використовувати JavaScript-бібліотеки та фреймворки безпосередньо, але з підтримкою статичної типізації та інших переваг TypeScript.

TypeScript знаходить широке застосування в розробці веб-додатків, особливо великих та складних проектів. Він покращує якість та підтримку коду, полегшує співпрацю між розробниками та дозволяє забезпечити більшу надійність та швидкість розробки програмного забезпечення.

1.3 Пакувальник Webpack

Webpack — це модульний стискач для JavaScript-програм, що використовується для збирання та управління залежностями веб-проекту. Він дозволяє розбити програму на модулі та імпортувати їх, а потім зібрати всі модулі в один або кілька пакетів, які можна використовувати на веб-сторінці[10]. Основні принципи роботи з Webpack:

— модульність: Webpack дозволяє використовувати модульну структуру веб-проекту, де кожен файл може бути модулем зі своєю функціональністю. Ви можете імпортувати модулі один в інший для зручного управління залежностями;

— залежності та завантаження: Webpack аналізує код вашого проекту та визначає всі його залежності. Він може завантажувати модулі з локальної файлової системи або зовнішніх джерел, таких як npm-пакети. Webpack автоматично вирішує залежності та забезпечує їх правильне завантаження під час виконання програми;

— збірка та оптимізація: Webpack забезпечує процес збірки проекту, який включає об'єднання всіх модулів в один або кілька пакетів. Він також здатен застосовувати різноманітні оптимізації, такі як мінімізація коду, оптимізація завантаження ресурсів, розділення коду на частини (code splitting) та багато іншого;

— завантаження зображень та інших активів: Webpack може бути налаштований для обробки різних типів файлів, таких як зображення, шрифти, таблиці стилів тощо. Він може автоматично перетворювати ці ресурси в формати, придатні для використання на веб-сторінці, і дозволяти їх завантаження відповідним чином;

— розширюваність: Webpack дозволяє розширити його функціональність за допомогою плагінів. Ці плагіни можуть виконувати різні завдання, від маніпуляції зі збіркою до автоматичної генерації HTML сторінок або інтеграції з іншими інструментами розробки.

Webpack має конфігураційний файл, в якому ви можете визначити правила для обробки різних типів файлів, вказати точку входу, налаштувати вивідні файли та багато іншого. Цей конфігураційний файл, зазвичай називається Webpack.config.js, дозволяє налаштувати Webpack під свої потреби.

Узагальнюючи, Webpack — це потужний інструмент для збирання та управління залежностями веб-проекту. Він допомагає забезпечити ефективну організацію та оптимізацію вашого JavaScript-коду, забезпечує швидку ініціалізацію веб-сторінки та полегшує розробку масштабованих додатків.

1.4 Бібліотека Babel для транспіляції коду

Babel — це популярна JavaScript-бібліотека, яка використовується для транспіляції (конвертації) сучасного синтаксису JavaScript до сумісного синтаксису, що підтримується старішими версіями браузерів та середовищами виконання JavaScript. Babel дозволяє розробникам використовувати нові функції та можливості JavaScript, навіть якщо браузери або середовища, в яких вони виконуються, не підтримують ці функції[11]. Основні особливості Babel:

— транспіляція сучасного синтаксису: Babel здатний перетворювати код, написаний з використанням нових функцій та синтаксису ECMAScript, на еквівалентний код, який може працювати в старіших

браузерах або середовищах виконання JavaScript. Це дозволяє розробникам використовувати останні можливості JavaScript без залежності від підтримки браузерами;

— плагінна архітектура: Babel має плагінну архітектуру, яка дозволяє розробникам використовувати різні плагіни для розширення можливостей транспіляції. Це дозволяє включати лише ті плагіни, які потрібні, і налаштовувати Babel під свої потреби;

— підтримка сучасних стандартів: Babel постійно оновлюється, щоб підтримувати нові версії стандарту ECMAScript та включати нові функції та можливості, які стають доступними. Це дозволяє розробникам використовувати останній синтаксис JavaScript навіть у старих браузерах;

— інтеграція з іншими інструментами: Babel добре інтегрується з іншими інструментами розробки, такими як Webpack, Gulp, Grunt та інші. Ви можете використовувати Babel як частину свого робочого процесу збирання та автоматизації, щоб автоматично транспілювати ваш код під час розробки.

Babel став популярним інструментом в сфері веб-розробки, особливо при роботі з сучасними фреймворками та бібліотеками JavaScript, такими як React, Angular, Vue.js та інші. Він дозволяє розробникам використовувати нові можливості мови, забезпечуючи при цьому максимальну сумісність з різними середовищами виконання.

1.5 Бібліотека RxJS для рективного програмування

RxJS (Reactive Extensions for JavaScript) — це бібліотека реактивного програмування для JavaScript, яка дозволяє простіше та потужніше робити асинхронні операції, керувати потоками даних та подій, а також реагувати

на зміни у них. Вона базується на концепції спостережуваних послідовностей (Observables) і надає набір операторів для маніпуляції даними та керування потоками подій[12]. Основні особливості RxJS:

— спостережувані послідовності (Observables): RxJS використовує поняття спостережуваних послідовностей, які представляють асинхронні потоки даних або подій. Спостережувані послідовності можуть видавати значення, помилки або сигнали завершення. Розробники можуть підписатися на спостережувану послідовність та реагувати на події, що виникають у цьому потоці;

— оператори: RxJS надає багатий набір операторів, які дозволяють маніпулювати та комбінувати спостережувані послідовності. Ці оператори дозволяють виконувати різні операції, такі як фільтрація, мапування, злиття, агрегація, затримка, обробка помилок та багато іншого. Вони дозволяють розробникам елегантно та декларативно маніпулювати потоками даних;

— асинхронність та конкуренція: RxJS дозволяє легко працювати з асинхронними операціями, такими як HTTP-запити, отримання подій від DOM, взаємодія з WebSocket і багато іншого. Вона надає інструменти для керування конкурентними операціями, обробки спільних ресурсів та синхронізації подій;

— підтримка архітектурних шаблонів: RxJS сприяє застосуванню різних архітектурних шаблонів, таких як спостерігач (Observer), ітератор (Iterator), функціональна програма, фасад та інші. Вона надає засоби для побудови реактивних компонентів, модулів та систем, які працюють разом з використанням спостережуваних послідовностей.

RxJS є потужною бібліотекою, яка дозволяє розробникам ефективно працювати з асинхронними операціями та потоками даних у JavaScript.

Вона знайшла широке застосування у розробці веб-додатків, особливо в сучасних фреймворках та архітектурах, де реактивний підхід дозволяє створювати більш ефективні та масштабовані програми.

3 КОНФІГУРУВАННЯ ФРОНТЕНД БІБЛІОТЕКИ

3.1 Налаштування середовища

Для встановлення Node.js була використана утиліта nvm, яка була завантажена з офіційного хітхабу наступною командою: curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.3/install.sh | bash

Після встановлення утиліти, потрібно було вибрати версію платформи. Враховуючи бажання зберегти підтримку бібліотеки, було обрано використовувати LTS(Long term support) версію Node.js. На даний час, номер цієї версії 18.16.0. Це стабільний реліз із довгим часом підтримки, що забезпечує гарантовану сумісність даної бібліотеки без переписування коду з майбутніми релізами платформи, а також сторонніми бібліотеками. Встановлюється вона з використанням утиліти nvm командою nvm install 18.16.0. Аби перевірити чи платформа встановлена, можна скористатися командою node -v, яка відразу покаже версію Node.js.

Наступним кроком було власне створення проекту та ініціалізація пакету залежностей та додаткової інформації. Це виконувалось шляхом запуску команди npm init[13]. Менеджер пакетів запитує детальну інформацію про проект, зокрема:

— package name — назва пакету (проекту);

— version — версія проекту;

— description — опис;

— entry point — вхідний файл програми;

— test command — команда для запуску тестів;

— git repository — адреса git-репозиторію;

— keywords — теги програми (використовуються для пошуку пакета);

— author — автор цифрового контенту;

— license — тип ліцензії.

Після внесення інформації, ми маємо змогу створити вхідний файл програми, який ми вказували в полі entry, та перевірити можливість його запуску командою npm start.

3.2 Завантаження залежностей

При роботі з проектами на Node.js залежності зазвичай завантажуються та зберігаються у папці node_modules. Докладна інформація про цей процес:

— завантаження залежностей:

1) залежності перераховуються у файлі package.json вашого проекту. Ви можете вказати їх у розділі dependencies або devDependencies, в залежності від того, чи ці залежності потрібні для продакшну або розробки;
2) коли ви запускаєте команду npm install у кореневій директорії проекту, npm переглядає вказані залежності у package.json і починає завантажувати їх з репозиторію npm;
3) залежності можуть бути завантажені з різних джерел, включаючи публічний репозиторій npm, приватні репозиторії, URL-адреси або локальні файлові шляхи.

— зберігання залежностей:

1) завантажені залежності зберігаються у папці node_modules вашого проекту. Кожен пакет залежності зазвичай має свою власну директорію в node_modules, названу відповідно до імені пакету;

2) у директорії кожного пакету зберігаються всі файли та папки, необхідні для його виконання. Це можуть бути JavaScript-файли, CSS-стилі, шаблони, зображення та інші активи;
3) крім того, для кожного пакету залежності у папці node_modules можуть бути наявні файли, які допомагають npm вирішувати залежності, такі як файл package.json пакету, README.md та інші метадані.

— версіонування залежностей:

1) кожна залежність може мати вказану версію у файлі package.json вашого проекту. Це дозволяє контролювати, яку версію конкретного пакету використовує ваш проект;
2) npm використовує Semantic Versioning (SemVer) для керування версіями пакетів. Це означає, що ви можете вказувати діапазон версій, наприклад, ^1.0.2, що означає, що ваш проект може використовувати будь-яку версію пакету, починаючи з 1.0.2, але не більше 2.0.0.

— рекурсивність залежностей:

1) алежності також можуть мати свої власні залежності. Коли ви встановлюєте пакет, npm автоматично встановлює його залежності та їх залежності, і так далі;
2) цей процес може бути рекурсивним, доки всі залежності не будуть встановлені та збережені у папці node_modules.

— виключення залежностей:

1) іноді можуть виникати випадки, коли ви хочете виключити певні залежності з установки або оновлення. Для цього ви можете використовувати опцію --no-save

або вручну видаляти папку певної залежності з node_modules.

Були встановлені наступні основні залежності (пакети для коректної роботи бібліотеки):

— typescript — утиліта для роботи з TypeScript-кодом;

— webpack — збирач проекту;

— babel-loader — транспілятор коду;

— rxjs — бібліотека реактивного програмування.

Крім цього, були завантажені плагіни та розширення для Webpack та Babel, про які детальніше розповідатиметься в наступних пунктах.

3.3 Налаштування середовища TypeScript

Крім встановлення, дана утиліта потребує правильного конфігурування. У кореневій директорії даного проекту було створено файл tsconfig.json. Цей файл містить налаштування компілятора TypeScript для проекту[14]. У файлі tsconfig.json потрібно визначити параметри компіляції. Основні параметри представлені нижче:

— компіляція:

1) target: Визначає цільову версію ECMAScript для компіляції вашого коду TypeScript;
2) module: Визначає модульну систему, яку використовуватиме ваш код, наприклад CommonJS, AMD, ES6 і т.д;
3) outDir: Вказує шлях до директорії, де будуть збережені скомпільовані JavaScript-файли;
4) rootDir: Визначає кореневу директорію вихідного коду.

— перевірка типів:

1) noImplicitAny: Вмикає строгий режим, що вимагає явного вказання типу для всіх змінних і параметрів;
2) strictNullChecks: Дозволяє перевіряти наявність значення null або undefined у змінних і параметрах;
3) strictFunctionTypes: Застосовує строгу перевірку типів при перевірці сумісності функцій;
4) noUnusedLocals та noUnusedParameters: Сигналізують про невикористані локальні змінні або параметри.

— пошук типів:

1) typeRoots: Вказує шляхи до папок, де розташовані визначення типів (.d.ts файлів);
2) types: Перелік пакетів, для яких ви хочете використовувати визначення типів.

— інші опції:

1) allowJs: Дозволяє компілювати JavaScript-файли разом з TypeScript-файлами;
2) esModuleInterop: Дозволяє використовувати імпорт та експорт у стилі ES модулів для не-ES модульних систем;
3) resolveJsonModule: Дозволяє імпортувати файли JSON як модулі.

Конфігурація для файлу tsconfig, що була обрана для даного проекту, продемонстрирована рисунку 3.1.

```
{
  "compilerOptions": {
    "target": "ESNext",
    "useDefineForClassFields": true,
    "module": "commonjs",
    "lib": ["ESNext", "DOM"],
    "moduleResolution": "Node",
    "strict": true,
    "resolveJsonModule": true,
    "isolatedModules": true,
    "esModuleInterop": true,
    "noEmit": true,
    "noUnusedLocals": true,
    "noUnusedParameters": true,
    "noImplicitReturns": true,
    "skipLibCheck": true,
    "outDir": "build",
    "allowJs": true,
    "baseUrl": "./",
    "paths": {
      "@jsx/*": ["./src/lib/jsx/*"],
      "@jsx": ["./src/lib/jsx/index"],
      "@reactive": ["./src/lib/reactive/index"],
      "@components": ["./src/lib/components/index"],
      "@statements": ["./src/lib/statements"],
      "@routing": ["./src/lib/routing"],
      "@common": ["./src/lib/common"],
      "@state-manager": ["./src/lib/state-manager"]
    },
    "jsx": "react-jsx",
    "jsxImportSource": "@jsx"
  },
  "include": ["src", "jsx"]
}
```

Рисунок 3.1 — Конфігурація tsconfig файлу

3.4 Використання Webpack та його плагінів

Написання фронтенду на бібліотеці, яка була реалізована передбачає розбиття коду на модулі. Аби не підключати кожен модуль програми вручну в HTML розмітку було використано утиліту Webpack, яка автоматично збирає весь проект в один вихідний файл — так званий, пакет.

пакет представляє собою мініфікований JavaScript-код. Тобто окрім зручності використання, ми отримуємо також зменшений розмір кінцевої програми.

Для правильного конфігурування утиліти в корні проекта було створено файл Webpack.config.js. Нижче продемонстрований опис основних параметрів конфігурації:

— context — директорія розміщення файлів;

— mode — тип генерування вихідного пакету (мініфікований для продакшину чи з коментарями для дев-розробки та відлагодження);

— entry — об’єкт, що містить вхідні точки для різних частин програми, що збиратимуться в пакети;

— output — об’єкт опцій для опису місцезнаходження та неймінгу вихідного файлу;

— plugins — плагіни, що використовуються для специфічної роботи утиліти;

— resolve — опис файлів, які будуть оброблені;

— module — додатковий специфічний функціонал обробляння конкретних файлів.

Користуючись тим, що Webpack підтримує розширення, було вирішено додати наступні плагіни для покращення досвіду користування кінцевого розробника:

— HTMLWebpackPlugin — автоматичне підключення вихідного пакет-файлу у заготовлений HTML;

— CleanWebpackPlugin — видалення застарілого пакету при генерації нових вихідних файлів програми;

— ProviderPlugin — використовується для автоматичного імпорту JavaScript-файлів. В контексті даної роботи додає в кожен файл функцію перетворення JSX-елементів в компоненти бібліотеки.

Крім того, що дана бібліотека уміє збирати вихідний код в єдиний пакет шляхом використання webpack, додаткове налаштування утиліти завдяки додаванню стороніх бібліотек в конфігураційну опцію module дозволило спростити імпортування статичних ресурсів а також підказки в шлясі імпорту. Приклад такого використання продемонстровано на рисунку 3.2.

```
3
4   import './styles.css'
5   import logo from '../../assets/logoYellow.png'
6
```

Рисунок 3.2 — Імпорт статичних ресурсів

Використання додаткових бібліотек style-loader та css-loader дає змогу імпорту css-файлів напряму в код. Можна використовувати декілька таких файлів, а бібліотеки та сконфігурована утиліта webpack зберуть їх в єдиний файл стилів.

Для імпорту зображень напряму в код була використана бібліотека file-loader та налаштована на роботу з наступними форматами файлів:

— PNG;

— SVG;

— JPG;

— GIF.

Для зручності запускі, а також аби уникнути проблеми CORS (Cross-Origin Resource Sharing) було підключено та додано утиліту devServer, яка автоматично піднімає сервер, який повертає HTML розмітку сайту.

3.5 Використання бібліотеки babel для транспіляції коду

В даному проекті, додатком до збирача webpack, використовується бібліотека для транспіляції коду babel. Перед створенням пакету вона перетворює код за певними правилами, які можна писати вручну або використовувати уже готові плагіни та збірки плагінів. Нижче зазначено плагіни та збірки для babel та типи транспіляції, що вони виконують в даному проекті:

— @babel/preset-typescript: TypeScript — JavaScript;

— @babel/plugin-transform-react-jsx: JSX — JavaScript;

— @babel/preset-env: JavaScript ES6+ — JavaScript ES5.

Для надійності коду бібліотека дозволяє писати код веб-застосунку на мові TypeScript. Утиліта webpack, під час створення пакету, запускає babel. Транспілятор, використовуючи плагін @babel/preset-typescript, перетворює код в JavaScript-файли.

Також для зручності, збільшення швидкості розробки і збереження схожості із найпопулярнішою бібліотекою для фронтенд розробки React, було вирішено використовувати розширений синтаксис JavaScript — JSX (JavaScript Extension). Це надає змогу використовувати в коді HTML розмітку так, ніби вона є частиною функціоналу. Приклад продемонстровано на рисунку 3.3. Цією частиною займається плагін @babel/plugin-transform-react-jsx.

```jsx
export const CharacterCard: FC<Character> = ({
  name,
  height,
  mass,
  hair_color,
  skin_color,
  eye_color,
  birth_year,
  gender,
}) => {
  return <div className="card-personagem">
    <h2>{name}</h2>
    <span>Height: {height / 100} m</span>
    <span>Mass: {mass} kg</span>
    <span>Hair Color: {hair_color}</span>
    <span>Skin Color: {skin_color}</span>
    <span>Eye Color: {eye_color}</span>
    <span>Birth Year: {birth_year}</span>
    <span>Gender: {gender}</span>
</div>
}
```

Рисунок 3.3 — Приклад JSX-коду

Останньою частиною є приведення JavaScript-коду до стандарту, що розуміють більшість браузерів. Так звані EcmaScript (ES) стандарти представляють собою набір версій різного функціоналу. Більшість сучасних браузерів розуміють JavaScript-код стандарту ES6 та вище. Старіші ж браузери, неправильно або геть не розуміти нових версій мови. Розробник, звичайно, бажає аби його застосунок працював на максимально великій кількості браузерів. Саме для цього відбувається пониження версії мови, шляхом транспіляції з використанням плагіну @babel/preset-env.

4 РЕАЛІЗАЦІЯ ФРОНТЕНД БІБЛІОТЕКИ

4.1 Структура проекту

Користуючись модульною системою, що надає платформа в поєднанні з утилітами, проект було розбито на логічні частини і поділено в наступну структуру папок:

— jsx — функція перетворення JSX-коду на компоненти бібліотеки;

— reactive — допоміжна обгортка реактивних об’єктів;

— components — логіка створення реактивних HTML елементів;

— statements — оператори для пізньої загрузки по умові;

— routing — утиліта для роутингу програми;

— state-manager — утиліта керування станом;

— common — спільні дані та типи.

Також, для зручного і зрозумілого імпорту файлів з бібліотеки, в файл конфігурації TypeScript додатково були додані скорочені псевдоніми шляхів.

4.2 JSX-runtime

Абстрагуючись від деталей транспіляції, перетворення JSX-коду є заміна компонентів, схожих на HTML тги на виклик певної функції. Аби розібратися як це працює, потрібно оглянути структуру HTML тегу, який буде перетворено[15]. Як приклад, оберемо some text . Даний компоненті містить наступні параметри:

— назва тегу: div;

— атрибути (props): { id: ”myDiv” };

— дочірні елементи(children): “some text”.

Знаючи компоненти тегу, можна зрозуміти, які параметри повинна приймати функція перетворення. Був створений файл jsx-runtime.ts з функцією перетворення jsx. назви та шлях були вказані в плагіні @babel/plugin-transform-react-jsx. Аби уникнути мануального імпорту, було використано webpack плагін ProviderPlugin, який на етапі збірки автоматично додає функцію jsx до кожного імпорту.

Функція jsx являє собою розподіляч, який перевіряє тип елементу, що приходить та, в залежності від типу, створює компонент бібліотеки одним із трьох способів, які будуть детально розглянуті в наступних пунктах:

— створення реактивного текстового компоненту;

— створення реактивного блочного компоненту;

— виклик функції компоненту.

Кінцевий результат транспіляції продемонстровано на рисунку 4.1.

```
// Before transpilation
const JsxComponent = <div id="myComponent">
  <p onclick={() => console.log('"p" was clicked')}>
    some text
  </p>
</div>;

// After transpilation
import { jsx } from "@jsx/jsx-runtime";
const TranspiledComponent = jsx(
  'div',
  { id: 'myComponent' },
  jsx(
    'p',
    { onclick(){ console.log('"p" was clicked') } },
    "some text"
  )
)
```

Рисунок 4.1 — Приклад транспіляції компонентів

4.3 Створення стандартних компонентів бібліотеки

На противагу бібліотеці React.js, яка використовує віртуальний DOM з постійним перевиконанням функцій компонентів для оновлення DOM, бібліотека, що представлена в даній роботі використовує реактивний підхід для оновлення HTML елементів. Компоненти кінцевої програми являють собою вложені функції, які викликаються лише один раз, на відміну від React.js. Такий підхід дозволяє обійтися без додаткових витрат процесорного часу для операцій бібліотеки, а також уникнути специфічних підходів оновлення та збереження даних компоненту, таких як React hooks.

Принцип оновлення кінцевих компонентів при зміні даних полягає в використанні патерну Observer. HTML елементи “підписуються” на зміни для певних даних і коли ці дані оновлюються система автоматично переписує лише ті компоненти, які залежать від цих даних.

Дану систему можна детально розібрати в функції, для створення реактивних текстових елементів, що продемонстрована на рисунку 4.2. Функція приймає назву тегу, його атрибути та дочірні компоненти. Спочатку створюється текстова DOM-вузол із ночірніх елементів, що були передані у функцію. Наступним кроком є створення кінцевого HTML елемент та додавання вузла до нього.

```
export const createTextComponent = <T extends keyof HTMLElementTagNameMap>(
  tag: T,
  options: Options<T>,
  children: Child[]
) => {
  const textNode = document.createTextNode(preparedString(children))
  const resultNode = document.createElement(tag)
  resultNode.appendChild(textNode);

  children.forEach((child, index) => {
    if (child instanceof Observable) {
      child.subscribe(value => {
        children[index] = value
        textNode.nodeValue = preparedString(children)
      })
    }
  })

  for (const key in options) {
    // @ts-ignore: Unreachable code error
    if (options[key] instanceof Observable) {
      // @ts-ignore: Unreachable code error
      options[key].subscribe(() => resultNode[key] = options[key].value)
    } else {
      // @ts-ignore: Unreachable code error
      resultNode[key] = options[key]
    }
  }

  return resultNode
}
```

Рисунок 4.2 — Створення реактивних текстових компонентів

Після створення, до елементи прикріплюється система для реактивного оновлення даних. Його дочірні елементи — це масив, який може містити примітивні типи, DOM-вузли, або реактивні дані. Функція проходить циклом по кожному із “дітей” та перевіряє їх тип. Якщо тип являє собою нащадком класу RxJS.Observable, компонент підписується на оновлення цих даних. При оновленні, компонент замінює дочірній реактивний елемент на його кінцеве значення та перезаписує текстовий вузол.

Крім реактивного оновлення вмісту елементів, також відбувається оновлення атрибутів. Система пробігається по кожному з атрибутів та перевіряє, чи є він нащадком класу RxJS.Observable. Якщо це реактивні дані, тоді компонент підписується на їх оновлення та передає функцію переписування атрибуту.

Для створення реактивних блочних компонентів використовується додаткова система для оновлення дочірніх елементів. Вони можуть приймати не лише нащадків класу RxJS.Observable, які містять примітивні значення, а й дані, які являються собою реактивні HTML елементи.

Функція також пробігається по кожному з “дітей” і підписується на них, якщо вони є реактивними. Лямбда-функція, що передається в метод subscribe, через інкапсуляцію інкапсуляції зберігає всі реактивні компоненти в окремий масив, пробігається по ньому, дістає HTML елементи з Observable та вставляє їх у відповідному порядку в батьківський компонент, що буде повернутий.

4.4 Створення користувацьких компонентів

Система дозволяє кінцевому розробнику створювати власні компоненти, тим самим розбиваючи частини коду, які можуть використовуватись повторно. Користувацьки компонент представляє собою функцію, яка повертає JSX-елемент.

Система використовує функцію jsx з модуля jsx-runtime для перевірки типу компонента, що був переданий. Якщо тип є функцією, це означає, що передано саме користувацький компонент і в такому випадку функцію буде виконано з передачею відповідних атрибутів, а також дочірніх елементів.

Для типізації користувацьких компонентів можна використовувати тип з представленої бібліотеки з назвою FC (Functional Component). Він є дженерік типом, який приймає опис атрибутів та дочірніх компонентів. Це дозволяє додати перевірку типів для власних JSX-компонентів.

4.5 Використання реактивних даних

Система дозволяє створювати реактивні дані засобами бібліотеки RxJS. Наприклад, використовуючи функцію timer можна створити Observable, який оновлюватиме число кожну певну кількість секунд. Після можна передати цей рективний об’єкт в компонент <p>. При старті проекту ми побачимо текстовий елемент, який автоматично оновлює числа.

Для простішого збереження та використання реактивних даних, а також для їх обміну, був створений невеличкий модуль state-manager для роботи із станом компонентів. Модуль повертає об’єкт $, який являє собою функцію, що приймає стандартне значення та на його основі створює екземпляр класу RxJS.BehaviorObject.

Крім цього об’єкт $ містить в собі додаткову функцію global для створення глобальних реактивних об’єктів, які можна використовувати в різних компонентах для обміну станом. Користувач має змогу створити реактивні дані в одному файлі, оновлювати ці дані в другому файлі, а редагувати на оновлення буде компонент, що знаходиться в третьому файлі.

Даний підхід схожий до маніпулювання станом в бібліотеці React.js (useContext hook) проте працювати з ним простіше адеж не потрібно постійно передавати об’єкт контексту.

4.6 Оператори пізньої загрузки

Як було сказано раніше, на відміну від бібліотеки React.js дана система виконує функції компонентів лише раз, перед відображенням сторінки. Такий підхід дає змогу економити ресурси але має нюанс: для відображення сторінки потрібно повністю згенерувати DOM-дерево.

З таким підходом виникає потреба у вирішенні наступних проблем:

— відображення компонентів за умовою;

— пізня загрузка елементів.

Для вирішення цих завдань було створено модуль statements, в якому представлені наступні функції:

— $when — головна функція, що приймає реактивні залежності, функцію умови відображення та компонент, який буде показуватися за виконанням умови;

— $if — функція схожа до попередньою але приймає лише одне реактивне значення, значення, яке задовольнятиме відображенню компонента та власне сам компонент;

— $preloadIf — схожа до попередньої. Різниця в тому, що вона завантажує одразу, а не при виконанні умови.

Для детальнішого розуміння пізньої загрузки по умові, пропоную розглянути логіку функції $when, що представлена на рисунку 4.3.

```
export const $when: FC<WhenProps> = ({
  values,
  satisfy,
  draw
}) => {
  const temp = <div/> as HTMLElement

  let parrentNode: Node
  let parrentIndex = 0
  let prevValue: any = undefined
  let child: HTMLElement

  combineLatest(values)
    .subscribe(async values => {
      if (!parrentNode) {
        await true
        parrentNode = temp.parentNode!
        parrentNode.childNodes.forEach((n, i) => n === temp && (parrentIndex = i))

        parrentNode.removeChild(temp)
      }

      const satisfyResult = satisfy(values)

      if (satisfyResult === prevValue) return
      prevValue = satisfyResult

      if (satisfyResult) {
        if (!child) child = draw(undefined)

        if (parrentIndex > parrentNode.childNodes.length) parrentNode.appendChild(child)
        else parrentNode.insertBefore(child, parrentNode.childNodes[parrentIndex])

      } else if (parrentNode.contains(child)) {
        parrentNode.removeChild(child)
      }
    })

  return temp
}
```

Рисунок 4.3 — Функція пізньої загрузки з умовою

Система створює елемент та одразу повертає його. Інші операції є асинхронними і будуть виконані після вмонтування даного компонента в DOM-дерево.

Використовуючи функцію combineLatest з бібліотеки RxJS для об’єднання реактивних даних, що передаються, створюється новий Observable. Функція підписки, що виконується при оновлені будь-яких даних, отримує батьківський вузол div-компонента, який ми вже повернули і який вмонтований в сторінку. Знаючи батьківський вузол, ми отримуємо індекс нашого компонента. Це потрібно аби при оновленні вставляти нові компоненти в правильне місце на сторінці. Батьківський вузол видаляється

з дерева. Це дозволяє писати розмітку без використання жодних системних елементів.

Після йде отримання результату виконання функції умови та перевірка чи її значення змінилось. Якщо значення нове і воно негативне, відбувається видалення компонента (якщо він присутній). В тому випадку, коли умова задовільнена, створюється компонент, який потрібно відобразити (якщо він не створений) та додається в DOM-дерево.

4.7 Роутинг

Ідея роутингу в фронтенд бібліотеках полягає у встановленні та управлінні шляхами (URL) веб-додатку. Це дозволяє створювати односторінкові додатки (Single Page Applications — SPA), де зміна URL не призводить до повного перезавантаження сторінки.

Для виконання було створено модуль routing, який використовує функцію $if для відображення компонета за умовою відповідності URL сторінки, до параметра, що був переданий. Реактивна залежність представляє собою об’єкт класу RxJS.BehaviorObject, який реагує на зміну параметра window.location.pathname.

Принцип роботи із системою роутингу продемонстровано на рисунку 4.4.

```
export const Routes = () => {
    return <div>
        <Route path="/" to={HomePage} />
        <Route path="/characters" to={CharactersPage} />
        <Route path="/planets" to={PlanetsPage} />
        <Route path="/starships" to={Starships} />
    </div>
}
```

Рисунок 4.4 — Використання роутингу

5 ОЦІНКА РОЗРОБЛЕНОЇ СИСТЕМИ

5.1 Створення веб-сайтів з використанням різних технологій

Оцінювати роботу системи варто порівнюючи із іншими фронтенд фреймворками або бібліотеками аби зрозуміти переваги переходу на дану технологію. Було створено два ідентичні сайти на різних технологіях: з використанням системи, що представлена в даній кваліфікаційній, і React.js, як бібліотеки, від якої відштовхнулась та покращила представлена система.

Тема сайтів — це невеличка бібліотека інформації про кіно-сагу “Зіркові війни”. Користувач може отримати інформацію про персонажів, кораблі та планети, згадані у фільмах.

Обидва сайти виконують HTTP-запити за допомогою бібліотеки axios до ресурсу SWAPI (Star Wars API), тобто використовують його, як бекенд. Виконання запитів повинне здійснюватись при переході на сторінку, а також при зміні віртуальної сторінки. В React-проект для цього були використані хуки (React hooks), сайт, що базується на бібліотеці, представленій в кваліфікаційній, виконує запити в функції підписки при оновлені реактивного об’єкту, що відповідає за номер віртуальної сторінки.

В обох застосунках присутній роутинг: React-сайт використовує для цього бібліотеку react-router-dom, дипломний сайт використовує вбудований модуль. Сайти містять наступні сторінки:

— домашня сторінка;

— опис планет;

— деталі про персонажів;

— характеристики космічних кораблів.

Обидва сайти використовують імпорт статичних ресурсів, а сема: декількох файлів стилів, растрових та векторних зображень.

5.2 Порівняння характеристик кінцевих сайтів

Однією з основних описаних проблем бібліотеки React було використання віртуальний DOM для оновлення DOM-дерева. Дана система тягне за собою додаткові витрати в об’ємі вихідного коду, процесорного часу та оперативної пам’яті.

Були створені пакунки обох версій веб-сайту. Для даного застосунку було визначено, що бібліотека представлена в кваліфікаційній роботі, потребує майже вдвічі менше пам’яті для зберігання програми, а саме: 110 КБ проти 209 КБ в порівнянні з React-аналогом, що продемонстровано на рисунку 5.1. Також час пакування був менший: 5.1 секунди проти 7.8 для аналога для бібліотеки React.js.

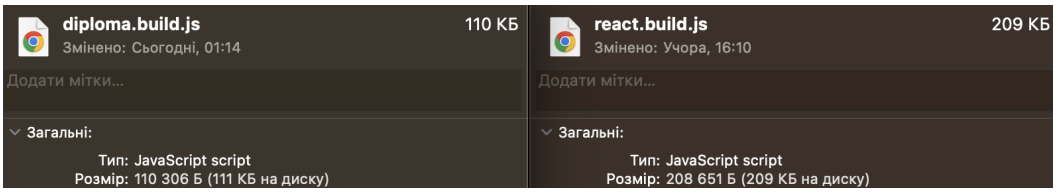

Рисунок 5.1 – Порівняння розмірів пакетів

За допомогою утиліт chrome dev tools було визначено максимальне використання операційної пам’яті для застосунків. Сайт на основі бібліотеки React мав найвищий показник використання пам’яті в межах 8.8 МБ, для системи на основі дипломної роботи, цей показник не перевищував 6.2 МБ, що говорить про більш ефективне використання оперативної пам’яті.

ВИСНОВКИ

У результаті роботи була розроблена власна фронтенд бібліотека з використанням транспіляції коду та реактивного оновлення HTML компонентів. При порівнянні програм, написаних з використанням даної бібліотеки та популярної альтернативи — React.js, виявлено наступні результати:

— використання власної бібліотеки дозволило досягти зменшення обсягу пам'яті, який займається програмою. Це свідчить про ефективність розробленої бібліотеки та її оптимізацію під вимоги роботи з фронтенд-застосунками. Крім того, програми, розроблені з використанням власної бібліотеки, продемонстрували кращу продуктивність та ефективність порівняно з програмами, що використовують React.js;

— у процесі розробки було налаштовано збирач, що дозволяє створювати єдиний мінімальний пакет застосунка. Це сприяє оптимізації завантаження та виконання програми та покращує її продуктивність;

— використання утиліти babel для транспіляції TypeScript та JSX коду дозволило розробляти програми на більш високому рівні абстракції, що полегшує розробку та підтримку коду. Це також забезпечує сумісність програм з різними браузерами та дозволяє використовувати сучасні функції JavaScript у проекті;

— створена реактивна модель оновлення HTML компонентів на основі бібліотеки RxJS забезпечує автоматичне оновлення відображення при зміні даних. Це спрощує процес розробки та забезпечує більш плавну та реактивну взаємодію з користувачем;

— додавання утиліт для роутингу та менеджменту стану полегшує навігацію по сторінкам та керування станом додатка. Це дозволяє розробникам зосередитися на логіці програми, забезпечуючи зручну організацію коду та високу стабільність;

— у процесі розробки був написаний однаковий сайт з використанням розробленої власної бібліотеки та React.js. Порівняння швидкості та розміру пакетів обох застосунків показало переваги використання власної бібліотеки, зокрема, більш оптимізоване використання процесорних ресурсів та менший обсяг файлів для завантаження.

Усі вищезазначені результати свідчать про досягнення мети роботи, а саме про створення власної фронтенд бібліотеки, яка забезпечує ефективну та оптимізовану розробку веб-додатків, зменшення обсягу пам'яті та поліпшення продуктивності порівняно з найпопулярнішою альтернативою. Результати дослідження можуть бути використані для подальшого розвитку та вдосконалення фронтенд розробки.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Mikowski M. Single Page Web Applications: JavaScript end-to-end / M. Mikowski, J. Powell., 2013. – 432 с.
2. The Best Guide to Know What Is React [Електронний ресурс]. – 2023. – Режим доступу до ресурсу: https://www.simplilearn.com/tutorials/reactjs-tutorial/what-is-reactjs.
3. Fain Y. Angular Development with TypeScript / Y. Fain, A. Moiseev., 2018. – 530 с. – (2).
4. What is the virtual DOM in React? [Електронний ресурс]. – 2022. – Режим доступу до ресурсу: https://blog.logrocket.com/virtual-dom-react/.
5. Why Virtual DOM is slower [Електронний ресурс]. – 2017. – Режим доступу до ресурсу: https://medium.com/@hayavuk/why-virtual-dom-is-slower-2d9b964b4c9e.
6. Casciaro M. Node.js Design Patterns / M. Casciaro, L. Mammino., 2016. – 777 с. – (2).
7. An introduction to the NPM package manager [Електронний ресурс] – Режим доступу до ресурсу: https://nodejs.dev/en/learn/an-introduction-to-the-npm-package-manager.
8. Node Version Manager – NVM Install Guide [Електронний ресурс]. – 2022. – Режим доступу до ресурсу: https://www.freecodecamp.org/news/node-version-manager-nvm-install-guide/.
9. Vanderkam D. Effective TypeScript: 62 Specific Ways to Improve Your TypeScript / Dan Vanderkam., 2019. – 261 с. – (1).

10. Vepsäläinen J. SurviveJS — Webpack 5: From apprentice to master / Juho Vepsäläinen., 2020. – 337 с.
11. Okoro A. How to Setup Babel in Node.js [Електронний ресурс] / Alvin Okoro – Режим доступу до ресурсу: https://www.freecodecamp.org/news/setup-babel-in-nodejs/.
12. Daniels P. RxJS in Action / P. Daniels, L. Atencio., 2017. – 352 с.
13. Setting up a Node development environment [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/development_environment.
14. The Definitive TypeScript 5.0 Guide [Електронний ресурс]. – 2023. – Режим доступу до ресурсу: https://www.sitepen.com/blog/update-the-definitive-typescript-guide.
15. Вступ до JSX [Електронний ресурс] – Режим доступу до ресурсу: https://uk.legacy.reactjs.org/docs/introducing-jsx.HTML.