КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп'ютерних наук та кібернетики
Кафедра математичної інформатики

Кваліфікаційна робота

на здобуття ступеня бакалавра
за спеціальністю 122 Комп'ютерні науки
на тему:

РОЗРОБКА І ОПТИМІЗАЦІЯ УНІВЕРСАЛЬНОГО ВЕБ-ІНТЕРФЕЙСУ ЗА ДОПОМОГОЮ БІБЛІОТЕК REACT І REDUX

Виконала студентка 4-го курсу
Семенюк Єлизавета Сергіївна
Науковий керівник:
професор, кандидат фіз.-мат. наук
Вергунова Ірина Миколаївна

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

Студент 

Київ – 2022

РЕФЕРАТ

Обсяг роботи 53 сторінки, 9 ілюстрацій, 19 таблиць, 21 джерело посилань.

ОПИС ЗАСОБІВ РОЗРОБКИ ВЕБ-ДОДАТКІВ, РОЗРОБКА ВЕБ-ІНТЕРФЕЙСУ, ОПТИМІЗАЦІЯ ВЕБ-ІНТЕРФЕЙСУ, ДОДАТОК ДЛЯ ІНФОРМАЦІЙНОГО СУПРОВОДУ, НАВЧАЛЬНИЙ ПРОЦЕС, СПЕЦИФІКАЦІЯ ТА РОЗРОБКА.

Об’єктом розроблення веб-інтерфейсу є пошук розв’язання задачі доступу студентів та викладачів до інформації про навчальний процес. Метою дипломної роботи є розробка універсального конкурентоспроможного веб-інтерфейсу, що забезпечить доступність актуальної інформації про навчальний процес для студентів та викладачів.

Для розробки веб-інтерфейсу був обрано мову програмування JavaScript та середовище розробки JetBrains WebStorm. Середовищем виконання виступає довільний браузер. Пристрої, для яких була проведена розробка не обмежуються конкретною ОС (операційна система) або розширенням екрану. Отже, розроблений додаток є універсальним.

У результаті виконання роботи було досліджено засоби розробки веб-додатків, а саме інтерфейсів для них. Розроблено додаток, основною задачею якого було відображення інформації про розклад для студентів та викладачів українських вищих навчальних закладів. Досліджено практики оптимізації швидкості роботи додатку та розроблено рекомендації щодо них.

Отриманий досвід дозволяє з легкістю оптимізувати розробку нових додатків, що здатні конкурувати за швидкістю та функціональністю з нативними. Розроблений додаток має велику практичну значущість, так як він забезпечує зручний доступ до розкладу студентам та викладачам, що значно спрощує навчальний процес.

ЗМІСТ

ВСТУП

Оцінка поточного стану об'єкта розробки. Індустрія розробки веб-додатків є достатньо незрілою, але незважаючи на це вона вже захоплює топові позиції за кількістю в масовому використанні, завдяки високому попиту в сфері послуг, бізнесу та маркетингу. По функціональності і ефективності веб-додатки наздоганяють та деколи перевершують нативні додатки. Масштабний перехід до сайтів і веб-додатків принесли мобільні мережі наступного покоління (маркетинговий 4G) з теоретичною швидкістю близько 300 Мбіт/с на викачування. Тому не дивно, що увага компаній, що розробляють програмне забезпечення, все більше спрямована на веб-розробку.

Легкість в розповсюдженні, невелика «вага» та просте встановлення, а отже і доступність в будь-який момент є важливими перевагами перед іншими типами додатків. Відсутність замкненості на визначену екосистему, що притаманно нативним додаткам, дозволяє користуватися веб-додатком на універсальній кодовій базі з довільного пристрою. Веб-додатки, маючи деякі недоліки в швидкодії і загальній функціональності, мають багато вагомих переваг перед своїми попередниками.

Актуальність роботи. Внаслідок незрілості сфери розробки веб-додатків кількість доступних інструментів мала, а найпопулярніші рішення постійно оновлюються. Тому актуальним є дослідження, що розкриє доступні інструменти та методи розробки. А дослідження методів максимізації продуктивності і легкості веб-додатків за рахунок оптимізації може додати ще більше конкурентоспроможності зі швидкими нативними додатками, тому є не менш актуальним.

Щоб якісно дослідити розробку і оптимізацію на практиці було розроблено сервіс в області інформаційного забезпечення процесу навчання,

так як нинішні способи донесення інформації є досить застарілими та незручними. Основні критерії зручності в даному контексті:

- високий рівень доступності інформації
- структурування даних для простоти сприйняття

Внесення змін, що відповідають сучасним вимогам, до способів донесення інформації за рахунок переваг веб-додатків є важливою складовою актуальності теми роботи.

Мета й завдання роботи. Метою кваліфікаційної роботи є розробка універсального веб-інтерфейсу, що забезпечить легкий доступ до актуальної інформації про навчальний процес для студентів, викладачів та інших працівників університету. Задля досягнення було створено список завдань:

– дослідити й описати наявні засоби розробки легких, швидких та придатних до масштабування веб-інтерфейсів;
– спроєктувати та розробити клієнт веб-додатку;
– розробити рекомендації щодо оптимізації роботи додатку;

Об'єкт, методи й засоби розробки. Об’єктом розробки веб-інтерфейсу є пошук розв’язання задачі доступу студентів, викладачів та інших працівників університету до актуальної інформації стосовно навчального процесу. Розробці передує дослідження можливих інструментів та засобів розробки на мові програмування JavaScript. Після завершення розробки проведене дослідження засобів розробки з використанням інформації з вільних джерел, більшість з яких стосується документації, та виокремлені рекомендації на рахунок оптимізації швидкодії додатку.

Для розробки веб-інтерфейсу було обрано мову програмування JavaScript з ліцензованим середовищем розробки JetBrains WebStorm. Середовищем виконання додатку виступає довільний браузер. Додаток є

універсальним, адже взаємодія з додатком може проводитись з будь-яких девайсів, що не обмежуються ОС та розширенням екрану.

Можливі сфери застосування. По-перше, виконана робота може знайти практичне застосування в процесі розробки та оптимізації нових веб-інтерфейсів, адже вона містить в собі багато корисних порад, що базуються на попередньо проведеному дослідженні. По-друге, дана робота виступає базовим та універсальним програмним рішенням для забезпечення доступу учасників навчального процесу до актуальної інформації для вищих навчальних закладів.

РОЗДІЛ 1. АНАЛІЗ ЗАСОБІВ РОЗРОБКИ ВЕБ-ІНТЕРФЕЙСІВ

1.1. Поняття веб-інтерфейсу

Веб-додаток – це розподілений додаток, в якому клієнтом виступає браузер, а сервером – веб-сервер. Браузер в даному випадку грає роль так званого тонкого клієнту – логіка додатку знаходиться на сервері, в той час як браузер відображає потрібну інформацію. Передача даних користувача між клієнтом і сервером проводиться за допомогою мережі. Оскільки клієнти ніколи не є залежними від ОС користувача, веб-застосунки є міжплатформенними сервісами. Ця перевага надає будь-якому веб-додатку універсальності, через що вони стрімко набрали популярність кінці 1990-х – початку 2000-х років. Однак в останні роки зросла тенденція у розміщенні тонких клієнтів на сервері, зміщуючи логіку застосунку на клієнтський додаток. Такі зміни обумовлені зростанням вимог до якості і швидкості роботи інтерфейсу веб-додатку. Саме клієнтську частину додатку – інтерфейс – і називають веб-інтерфейсом.

1.2. Класифікація засобів розробки

Щоб визначити області пошуку згадаємо ключові поняття у розробці додатків. В першу чергу найважливішою частиною розробки програмного продукту є його архітектура. Від неї залежить як можливість розробити додаток так і його придатність до масштабування.

По-друге складно уявити зручний у використанні додаток, який не мав би графічного інтерфейсу. Навколо роботи за інтерфейсом користувача зосереджено багато практик, але більшість з них поступово втрачає актуальність з появою нових пристроїв та способів взаємодії. Наприклад, поява смартфонів і взаємодія з інтерфейсами за допомогою жестів та голосу.

На останок, для пришвидшення розробки і підвищення якості виконуваної задачі очевидною є користь від використання допоміжних бібліотек. Вони накопичують у собі набір напрацювань в сфері розробки, що вирішують свою задачу найкращим чином. В такому випадку впровадження самотужки розроблених бібліотек найчастіше є надлишковим і неоптимальним. Бібліотеки можуть допомагати вирішити проблему обробки даних, тестування та впровадження додатків.

Підсумовуючи згадану вище інформацію, виділимо такі категорії досліджуваних засобів розробки:

– архітектурні патерни;
– засоби роботи з інтерфейсом;
– допоміжні бібліотеки.

1.3. Архітектурні патерни

1.3.1. Model-View-Controller

Model-View-Controller, або MVC – патерн, що використовується для розробки програмного забезпечення і розподіляє його на три взаємопов'язаних частини (рисунок 1). Це робиться для відокремлення внутрішнього представлення інформації від способів подання та отримання її користувачами [1]. Відокремлення даних компонентів у шаблоні проєктування MVC дозволяє паралельно розробляти та ефективно використовувати код.

Традиційно цей патерн використовується для графічних інтерфейсів користувача. Також ця архітектура стала популярною для розробки веб-додатків і навіть мобільних додатків.

Компоненти в MVC архітектурі:

– model(модель) – найважливіший компонент патерну, що виражає поведінку програми з точки зору предметної області, незалежно від

користувацького інтерфейсу [2]. Він безпосередньо керує даними, логікою та правилами перетворення;

– view(вигляд) – візуальна інтерпретація даних, як то графік, або діаграма. Патерном передбачається кілька інтерпретацій одних і тих же даних;
– controller(контролер) – приймає на вхід дії користувача та конвертує їх у команди для моделі, або вигляду [3].

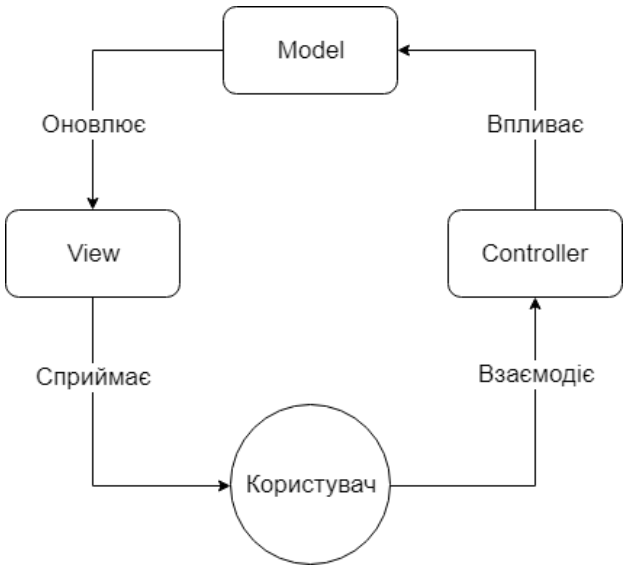


Рис. 1 – Три архітектурних компоненти в MVC

На додаток до поділу програми на три види компонентів, конструкція MVC визначає взаємодію між ними [4]:

– model – відповідає за управління даними програми. Вона отримує вхідні дані користувача від контролера;
– view – представляє модель в певному форматі.
– controller – реагує на вхідні дані користувача та запускає процес обробки даних в моделі. Контролер отримує вхідні дані, за бажанням виконує їх валідацію, а потім передає їх до моделі.

У ранні етапи розробки в Інтернеті архітектура MVC була в основному реалізована на стороні сервера, вимагаючи оновлення клієнту за допомогою форм або посилань, та отримання оновлених сторінок назад для відображення в браузері. Зараз більша кількість логіки переміщується на клієнт з появою та швидким розвитком інформаційних сховищ на стороні клієнта. В той час як XMLHttpRequest – спосіб взаємодії з сервером без перезавантаження сторінки – дозволяє періодично оновлювати дані за необхідності [5].

Популярні веб-фреймворки, такі як AngularJS, Ember.js та Backbone, реалізують архітектуру MVC, хоча і трохи адаптованому вигляді.

1.3.2. Model-View-Presenter

Model-View-Presenter (MVP) – це відгалуження архітектурного шаблону MVC, що використовується переважно для побудови користувацьких інтерфейсів. Патерн також поділяє систему на три частини: model (модель), view (вигляд) і presenter (представник) (рис. 2). У MVP представлення бере на себе обов'язки проміжного ланцюга між моделлю і виглядом. Вся логіка вигляду переноситься до представлення [6].

Даний шаблон проєктування UI (user-inteface) був розроблений для полегшення автоматизованого модульного тестування та покращення розділення відповідальності в логіці відображення (розділення логіки від вигляду). Патерн містить такі складові:

- модель – зосереджує в собі бізнес-логіку, за необхідності підтягує дані зі сховища;
- вигляд – реалізує відображення даних, отриманих з моделі, але оновлення проходять через представника;
- представник відповідає за реалізацію взаємодії між моделлю та виглядом.

Зазвичай екземпляр вигляду створює екземпляр представника, передаючи йому посилання на самого себе. Представник(presenter) взаємодіє з виглядом(view) в абстрактному вигляді, через його інтерфейс. Коли з’являється подія у вигляді, викликається відповідний метод представника, який не містить у собі ні параметрів, ні значення, що повертається. Отримання необхідних даних про поточний стан інтерфейсу представником відбувається через інтерфейс вигляду і через нього ж дані з моделі(model) та інші результати роботи передаються у вигляд.

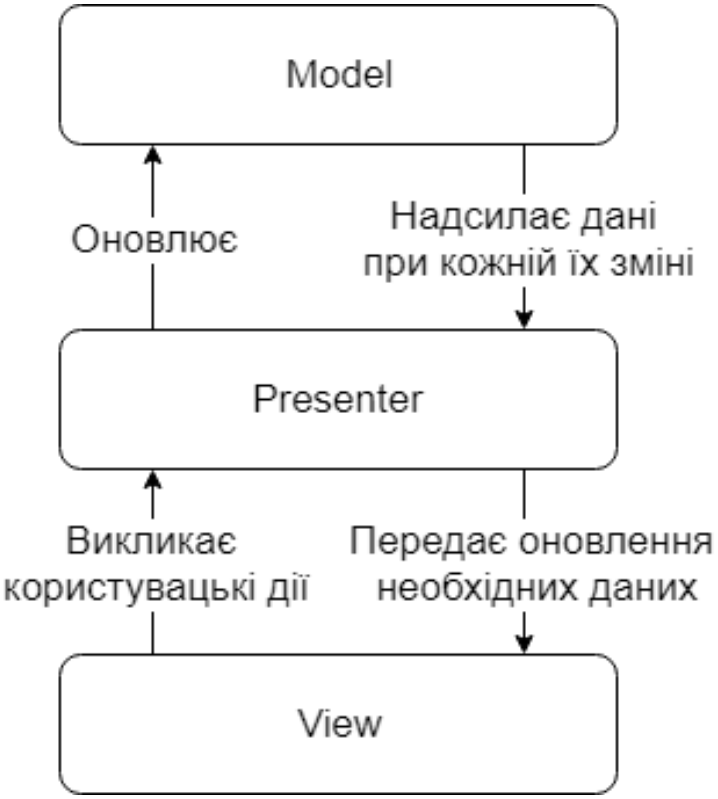


Рис. 2 – Три архітектурних компоненти в MVP

1.3.3. Flux

Flux – інтерпретація MVC патерну, що запропонована розробниками компанії Facebook. Патерн радить використовувати таку структуру додатку, щоб складові частини утворили односторонній потік даних. За словами творців принципу таке правило дозволить полегшити підтримку і масштабування проєкту. Щоб зрозуміти, як працює Flux, опишемо і пояснимо призначення кожної з його основних складових:

- actions – допоміжні методи, що реалізують передачу даних до dispatcher
- dispatcher – отримують actions та передають дані до зареєстрованих функцій зворотнього виклику;
- store – контейнер що зберігає стан та логіку додатку;
- view – React компоненти що отримують стан зі store і передають їх дочірнім компонентам через параметри.

Опишемо роботу компонентів патерну (рис. 3). Всі дані проходять через dispatcher як центральний вузол. Дії передаються до dispatcher через генерацію «розробника дій» і найчастіше вони э результатом взаємодії користувачів із view. Далі dispatcher виконує оновлення store, відправляючи дії до нього. Кожен store реагує на дії, що відповідають його предметній області. Після цього store генерують подію зміни, щоб сповістити view, що відбулася зміна даних. View слухають ці події та отримують дані зі store у обробнику подій. Далі view запускають процес генераціі нового вигляду, базуючись на отриманих даних [7].

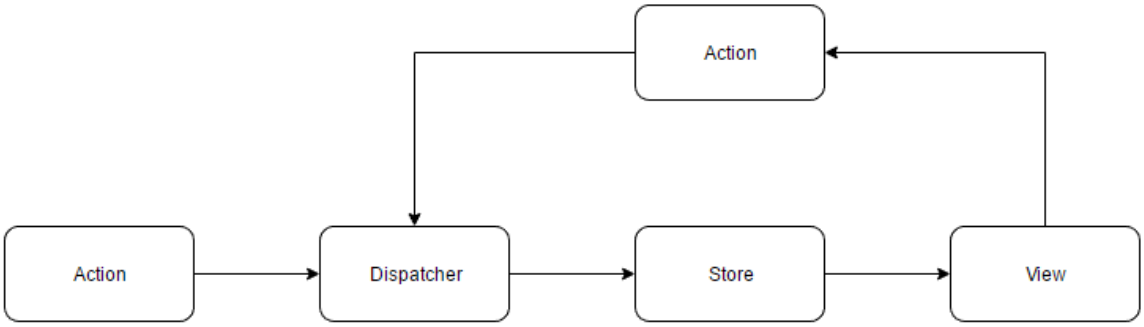


Рис. 3 – Діаграма компонентів патерну Flux

Стан програми зберігається виключно в store, через що різні частинам програми мають можливіть залишатися відокремленими. Залежності між store зберігаються в суворій ієрархії, а синхронні оновлення керуються dispatcher.

Виявлено, що двостороння прив'язка даних призводить до каскадних оновлень, коли зміна одного об'єкта зумовлює зміну іншого об'єкта, що може викликати ще більше оновлень в майбутньому. Зі зростанням функціональності додатку результат взаємодії з користувачем стає дуже важко спрогнозувати, в наслідок цих каскадних оновлень. Коли оновлення можуть змінюють дані лише за одну ітерацію, система в цілому стає більш передбачуваною. Цим пояснюється перевага одностороннього потоку даних.

Даний патерн активно використовується в екосистемі React і має достатню кількість прикладів успішного впровадження в розробку веб-додатків.

1.4. Засоби роботи з інтерфейсом користувача

Для вибору найбільш використовуваних засобів розробки інтерфейсів було прийнято рішення проаналізувати кількість користувачів для кожного з

них. Основним джерелом даної інформації є репозиторій пакетів – NPM. Це центральне сховище усіх javscript бібліотек, яке використовують для їх зберігання і поширення. Тому якщо зібрати інформацію про завантаження з цього репозиторію, то можна отримати достатньо репрезентативні дані про популярність інструментів.

Дослідження проводилося серед таких бібліотек:

- React;
- Preact;
- Vue;
- Ember;
- Angular 1;
- Angular 2;
- Angular;
- Backbone.

На графіку приросту завантажень відносно всіх бібліотек (рис) видно очевидних фаворитів. Лідируючі позиції займає React, наступними йдуть фремворки Angular різної версії. Серед всіх бібліотек вирізняється зростаючий інтерес до бібліотеки Vue. Проведемо короткий опис даних бібліотек у наступних пунктах.

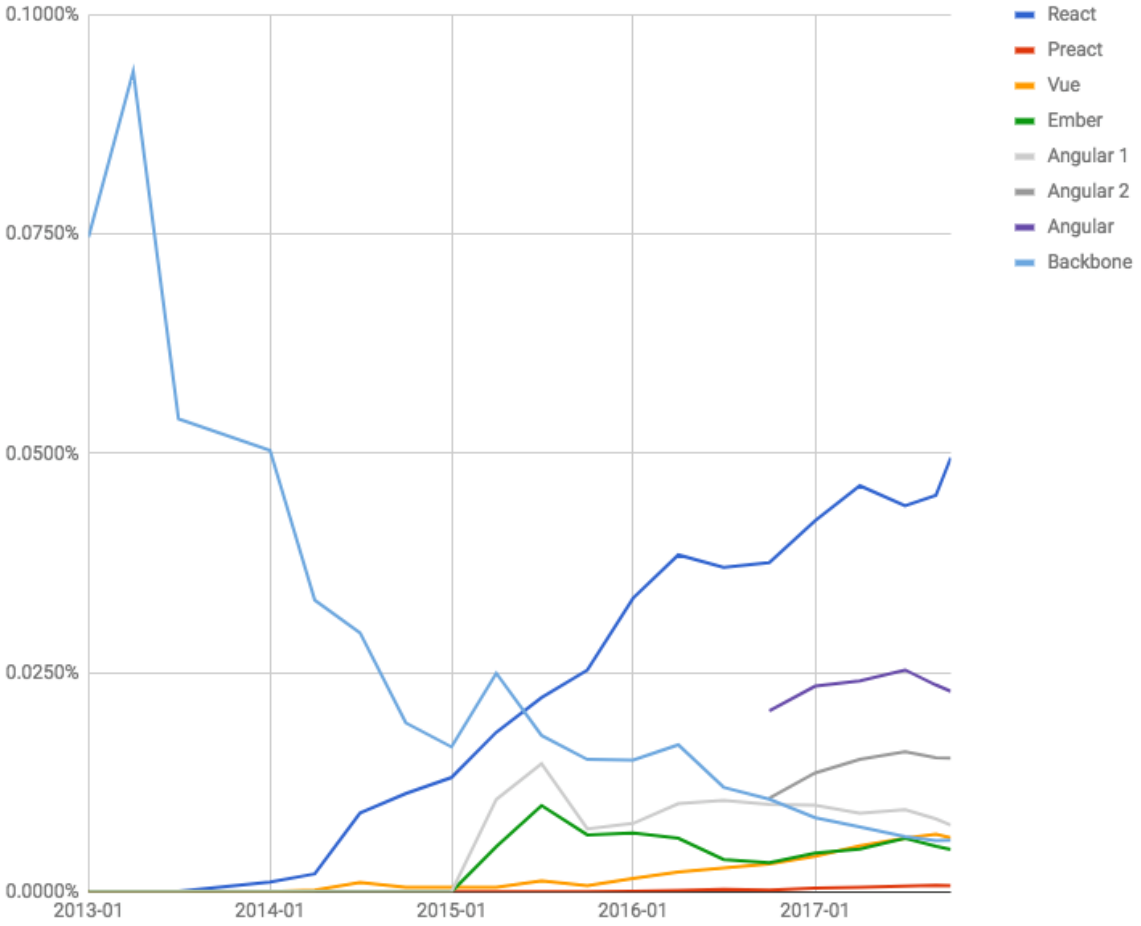


Рис. 4 – Графік приросту завантажень з репозиторію NPM

1.4.1. Фреймворк Angular

Angular – JavaScript-фреймворк з відкритим програмним кодом, що розробляється та оновлюється компанією Google. Призначений для розробки односторінкових додатків, що містять лише HTML сторінку з CSS і JavaScript. Головна мета – розширення браузерних застосунків за основою темплейту Model-View-Controller (MVC) та полегшення процесу розробки й тестування. Основна інформація про фреймворк наведена в таблиці 1.

Фреймворк працює з HTML-сторінкою, що містить додаткові атрибути і пов'язує області вводу або виводу сторінки з моделлю, яка є звичайними

змінними JavaScript. Значення цих змінних задаються вручну або отримуються зі статичних або динамічних JSON-даних.

За даними служби аналізу JavaScript для Libscore, зараз AngularJS входить до трійки сучасних проєктів, що набрали найбільшу кількість зірок на GitHub [8].

Таблиця 1 – Основна інформація про Angular

Назва параметру	Значення
Тип	JavaScript фремворк, односторінковий застосунок
Розробник	Google Inc. та спільнота
Розмір	144 кілобайти
Стан розробки	Активний
Ліцензія	MIT License

AngularJS побудований на переконанні, що декларативне програмування слід використовувати для створення користувацьких інтерфейсів та підключення компонентів програмного забезпечення, тоді як імперативне програмування краще підходить для визначення бізнес-логіки додатка. Фреймворк адаптує та розширює традиційний HTML, щоб представити динамічний вміст через двостороннє зв'язування даних, що дозволяє автоматично синхронізувати моделі та перегляди. Як результат, AngularJS зменшує значення явної DOM-маніпуляції з метою покращення тестування та продуктивності.

Основні цілі AngularJS включають:

– відокремлення DOM-маніпуляцій від логіки додатків, що суттєво впливає на спосіб побудови коду;

– розділення клієнта та сервера, що дозволяє працювати паралельно і використовувати повторно як клієнт, так і сервер;
– проведення розробника через повний шлях розробки застосунку: від проєктування UI (User Interface), через написання бізнес-логіки, до останньої стадії - тестування.

AngularJS використовує темплейт MVC для відокремлення представлення, даних та логічних складових. Angular постачає сервісні служби залежні від вигляду (наприклад, контролери) для клієнт-серверних веб-додатків за допомогою впровадження залежності. Відповідно, навантаження на сервер значно зменшується.

1.4.2. Фреймворк Vue.js

Vue.js (українською «в'ю», від англ. view) – JavaScript-фреймворк що використовує шаблон MVVM для створення інтерфейсів користувача на основі моделей даних [9], через реактивне зв'язування даних. Основна інформація про фреймворк наведена в таблиці 2.

Vue використовує синтаксис темплейтів на основі HTML, що дозволяє декларативно по'язувати рендеринг DOM з основними екземплярами даних в Vue. Всередині Vue компілює шаблони в функції віртуального DOM. В поєднанні з реактивною системою, він здатний обрахувати та визначити необхідну для повторного рендерингу кількість компонентів, після чого провести мінімальну кількість маніпуляцій з DOM, як тільки стан додатку зміниться.

Таблиця 2 – Основна інформація про Vue

Назва параметру	Значення
Тип	JavaScript бібліотека
Розробник	Автор - Evan You, продовжує розробку спільнота
Розмір	30.67 кілобайти
Стан розробки	Активний
Ліцензія	MIT License

В Vue ви можете використовувати синтаксис шаблонів або писати рендерингові функції використовуючи JSX. Функція рендерингу відкриває можливості для потужних патернів, створених на базі компонентів.

Одна із найвиразніших особливостей Vue – це ненав'язлива реактивна система. Моделями є прості JavaScript об'єкти, що робить керування станами доступним та інтуітивним. Vue дає можливість оптимізованого повторного рендерингу, що не вимагає жодних додаткових налаштувань. Кожен компонент несе відповідальність за свої реактивні залежності під час рендерингу. Це дозволяє системі точно визначити ідеальний момент для ре-рендерингу компонентів, які його потребують. Цей фреймворк широко використовується компанією Gramarly – це український стартап, що відносно нещодавно вийшов на міжнародний ринок.

1.4.3. Бібліотека React

React – відкрита JavaScript бібліотека для реалізації UI, яка створена для розв’язання проблем часткового оновлення вмісту веб-сторінки, з якими часто стикаються під час розробки односторінкових додатків. Розробляється великими компаніями, такими як Facebook або Instagram та розробниками-індивідуалами [10, 11, 12]. Інформація про бібліотеку наведена в таблиці 3.

Таблиця 3 – Основна інформація про React

Назва параметру	Значення
Тип	JavaScript бібліотека
Розробник	Facebook, Instagram та спільнота
Розмір	128 кілобайт
Стан розробки	Активний
Ліцензія	MIT License

React надає розробникам можливість створювати веб-застосунки, які використовують дані, котрі оновлюються, без необхідності перезавантаження сторінки. Його мета полягає в тому, щоб бути швидким, простим, масштабованим. ReactJs обробляє виключно UI у додатках, що відповідає складовій View у шаблоні Model-View-Controller (MVC). Як бібліотека для UI React часто використовується разом з іншими бібліотеками, такими як Redux. Зараз React активно використовують такі компанії як Netflix, Yahoo [13], Airbnb [14] та інші.

React не намагається надати повну "схему додатків". Він безпосередньо спрямований на побудову UI, тому вважається класичним базовим інструментом та не містить в собі інші, які деякі розробники вважають необхідними для створення повноцінної програми. На мою думку, це є значною перевагою React, так як він не обмежує розробника та надає йому повну свободу вибору будь-яких надбудов, котрі вважаються кращими для виконання певних завдань (наприклад, існують бібліотеки для реалізації доступу до мережі або локального зберігання даних).

Для підтримки одностороннього потоку даних, що є концептуальною особливістю React, використовується архітектура Flux, що згадувалася у попередньому пункті. React під архітектурою Flux не зобов'язаний змінювати жодні передані ззовні дані. Натомість він може передати функції зворотнього виклику, що створюють дії, які диспетчер надсилає задля модифікації сховища. Сховище, в свою чергу, змінюється у відповідь на дії, отримані від диспетчера.

Багато реалізацій Flux створено з моменту його винайдення. У своїй роботі я надаю перевагу одній з найпопулярніших – Redux. Він має одне сховище, ключовою особливістю якого є практика структурування інформаційних моделей таким чином, щоб кожен окремий елемент даних зберігається рівно один раз. У деяких джерелах така практика носить цікаву назву «єдине джерело правди», що вдало відображає основну концепцію.

1.5. Допоміжні бібліотеки

Розробка додатків, як ми вже визначили, може займати в декілька разів більше часу, якщо не використовувати бібліотеки. Вони допомагають швидко виконати спеціалізовані задачі не поступаючись якістю спроєктованому коду. Наведемо найпоширеніші з них.

1.5.1. Робота зі станом додатку

Redux – відкрита JavaScript бібліотека призначена для управління станом програми. Найчастіше використовується як надбудова до React або разом з Angular для побудови користувацьких інтерфейсів.

Стан програми зберігається в деревовидній структурі об'єктів в одному сховищі. Насамперед, це дозволяє зберігати стан додатка в процесі роботи над

ним, що значно пришвидшує сам цикл розробки. Очевидно, що одне дерево станів значно полегшує рефакторинг та тестування програми.

Єдиний спосіб змінити стан – це виокремити дію, об'єкт, що описує останню зміну. Це гарантує, що перегляди або виклики мережі можуть лише виражати намір змінити поточний стан, але ніколи не зможуть це зробити. Всі зміни є повністю централізованими та послідовними. Оскільки дії є простими об'єктами, розробник має можливість зберігати та структурувати їх. Це може бути корисним для подальшого рефакторингу або тестування.

MobX – бібліотека, що протестована в бойових умовах. Вона спрощує та маштабує керування станом, прозоро застосовуючи функціональне реактивне програмування. Основна концепція MobX звучить так: «Якщо щось має бути отримано зі стану додатку, це обов’язково буде отримано». Це включає в себе користувацький інтерфейс, серіалізацію даних, зв'язок з сервером тощо [15].

1.5.2. Перетворення даних

Reselect – бібліотека для створення запам’ятовуючих селекторов (memoized selectors). Ми визначаємо селектори, як функції, що витягають фрагменти стану для компонентів. Використовуючи запам'ятання, можна запобігти зайвий рендеринг і перерахунок отриманих данних, що в свою чергу пришвидшить наш додаток.

Ramda – багатофункціональна функційна JavaScript бібліотека. Основними відмінними рисами Ramda є:

- чистий функціональний стиль. Імутабельні та вільні від побічних ефектів функції є основою даної бібліотеки. Це може допомогти надати коду елегантного та простого вигляду;

– функції Ramda за замовченням використовують каррінг. Це дозволяє легко створювати нові функції зі старих, просто не надаючи кінцевих параметрів.
– параметри для функцій Ramda розташовані так, щоб зробити їх зручним для каррінгу. Дані, які будуть відомі найпізніше зазвичай надаються останніми.

Lodash – бібліотека форматування даних. Вона робить програмування на JavaScript легше за рахунок взяття на себе рутинної роботи з масивами, числами, об’єктами, рядками тощо. Ця бібліотека знаходить своє застосування у таких задачах:

– ітерація масивів, об’єктів, рядків;
– маніпуляція та тестування значень;
– створення складних функцій за рахунок композиції.

1.5.3. Тестування

Jest – бібліотека тестування, розроблена компанією Facebook. Вирізняється своєю простотою та працює без необхідності додаткового конфігурування. Підходить для тестування проєктів на базі бібліотеки React. Додатковою особливістю є можливість знімкового тестування, яке вирішує проблему пошуку небажаних змін впродовж розробки проєкту.

Enzyme – бібліотека, що дозволяє просто тестувати складні React-компоненти. Головна її перевага в тому, що вона може виконувати так званий поверхневий рендеринг – коли компонент тестується поверхнево, не зачіпаючи тих, від яких він залежить.

Mocha – це фреймворк для тестування коду, написаного мовою JavaScript, що працює на основі node.js, підтримує роботу з довільними

браузерами, асинхронне тестування, звіти про покриття продукту тестами, а також роботу з будь-якими бібліотеками припущень.

Chai – бібліотека припущень, що може працювати з будь-яким тестовим фреймворком. Бібліотека використовується при розробці за технікою TDD (Test-driven-development). Може виконуватись як у браузері так і у середовищі Node.js.

РОЗДІЛ 2. ПРОЄКТУВАННЯ ТА РОЗРОБКА

2.1. Розробка вимог

Реалізація ПЗ – це складний процес перекладу системної специфікації в цілісну працездатну систему. Етап реалізації містить процеси проєктування і програмування. При застосуванні еволюційного підходу, перший етап також може вимагати фундаментального змінювання системної специфікації [16].

На етапі проєктування ПО визначається його структура, дані, які є частиною системи, інтерфейси взаємодії компонентів системи і іноді алгоритми, що мають використовуватись. Проєктувальники відразу ніколи не отримують закінчений результат – процес проєктування зазвичай проходить через розробку кількох проміжних версій ПЗ. Проєктування передбачає послідовну формалізацію і деталізацію створюваного ПО з можливістю внесення змін до рішень, прийнятих на більш ранніх стадіях проєктування.

Процес проєктування може включати розробку декількох моделей системи різних рівнів узагальнення. Оскільки проєктування є по суті процесом декомпозиції, такі моделі допомагають виявити помилки, зроблені на ранніх стадіях проєктування, а отже, дозволяють внести зміни в раніше створені моделі в майбутньому.

Результатом кожного етапу проєктування є специфікація, необхідна для виконання наступного етапу, що поступово стає все більш деталізованою. Кінцевими результатами процесу проєктування є точні специфікації на алгоритми і структури даних, що в подальшому будуть застосовані на наступному етапі створення ПО.

2.1.1. Опис користувачів

Формалізацію вимог варто почати з визначення користувачів системи. Оскільки додаток розроблюється для забезпечення інформаційним супроводом вищих навчальних закладів, то основними користувачами в системі повинні бути:

- студент – особа, що навчається в університеті. Характеризується назвою ВНЗ, факультетом, спеціальністю та роком навчання. За спеціальністю та роком навчання визначається розклад пар та графік сесійного контролю;
- викладач – особа, що викладає в університеті. Може викладати пари на різних факультетах. Характеризується назвою ВНЗ та прізвищем, ім’ям і по-батькові.

2.1.2. Сценарії використання

Для точнішого визначення цільового продукту необхідно прорахувати та окреслити сценарії його використання. Нехай ми маємо декілька можливих сценаріїв. Для кожного складемо окрему таблицю, в якій опишемо діючих осіб, їхню взаємодію з продуктом та основні вимоги у неофіційному форматі.

По-перше, для повноцінного доступу до розкладу занять на деякому факультеті потрібно реалізувати можливість розкладу студента (табл. 4) та розкладу викладача (табл. 5). Пошук має працювати таким чином, щоб його було зручно здійснювати на постійній основі. Одним із можливих рішень для майбутнього проєктування дизайну сторінки веб-застосунку рекомендованано додання кнопок у вигляді посилань на розклади викладача та студента на головну сторінку.

Таблиця 4 – Сценарій «Пошук розкладу студента»

Користувач	Вимоги
Студент	Для початку користувачу потрібно обрати опцію «Розклад студента». Для того, щоб знайти розклад у системі ввести місце навчання – університет, факультет та групу. Після отримання даної інформації відбувається пошук розкладу у системі. У разі успіху користувачеві відображується розклад. У разі невдачі – відображається сторінка з повідомленням «розклад не знайдено» і пропозиція перейти назад до пошуку.

Таблиця 5 – Сценарій «Пошук розкладу викладача»

Користувач	Вимоги
Викладач, студент	Для початку користувачу потрібно вибрати опцію «Розклад викладача». Для відображення розкладу викладача з системи – ввести університет, в якому викладає, та знайти потрібне ПІП зі списку викладачів, що викладають в цьому університеті.

Після знайдення розкладу з’являється доступ до інтерфейсу розкладу студента (табл. 6) або викладача (табл. 7).

Таблиця 6 – Сценарій «Перегляд розкладу студента»

Користувач	Вимоги
Студент	Розклад пар повинен відображатися на весь тиждень, окрім вихідних. Дні тижня розташовуються вздовж горизонтальної осі. Пари на кожен день – під назвою дня

	тижня вздовж вертикальної осі. За замовчуванням для кожної пари повинна бути доступна така інформація: – день тижня; – час проведення; – аудиторія та корпус; – назва предмету; – ПІП викладача; – тип пари (лекція, семінар, лабораторна) – номер підгрупи, якщо є – парність тижня, якщо є

Таблиця 7 – Сценарій «Перегляд розкладу викладача»

Користувач	Вимоги
Викладач, студент	Розклад пар повинен відображатися на весь тиждень, окрім вихідних. Дні тижня розташовуються вздовж горизонтальної осі. Оскільки час проведення пар на різних факультетах може відрізнятися, тому пари повинні відображатись на шкалі часу, вздовж вертикальної осі. Таким чином, щоб розташування пари відповідало часу проведення пари. Необхідна для відображення інформація: – час проведення; – назва предмету; – аудиторія та корпус;

	– факультет та група; – номер підгрупи; – парність тижня;

Щоб у розкладі розмістити лише необхідну інформацію про пари і не засмічувати інтерфейс надлишковими даними потрібно передбачити вікно детального перегляду інформації (табл. 8).

Іншою важливою інформацією стосовно навчального процесу є розклад семестрового контролю (табл. 9), який обов’язково повинен мати інформацію про заліки, екзамени, захисти робіт та консультації.

Таблиця 8 – Сценарій «Перегляд подробиць про пару»

Користувач	Вимоги
Студент	Для перегляду деталізованої інформації про пару в окремому вікні необхідно натиснути на пару в розкладі. Для перегляду, окрім основної інформації повинні бути доступні такі подробиці: – кількість пар, що пройшла; – кількість пар, що залишилась; – дати проведення пар; – детальний час проведення пари з перервами; – номер пари;

	– фотографія викладача.

Таблиця 9 – Сценарій «Перегляд графіку семестрового контролю»

Користувач	Вимоги
Студент	Графік семестрового контролю доступний після введення назви ВНЗ, факультету та групи. Графік повинен мати вигляд календаря, в якому буде розміщуватись інформація про контроль, що проходить в кожен з днів. Необхідна для виводу інформація: – назва контролю; – тип контролю (екзамен, залік, захист, консультація); – дата проведення; – час проведення; – аудиторія; – викладацький склад;

Для зручності користування система повинна передбачити спосіб швидкого переходу до потрібного розкладу за рахунок використання посилань (табл. 10). Це заощадить час користувача, позбавивши його від надлишкових дій

Для справного неперервного забезпечення актуальною інформацією було передбачено функцію повідомлення модераторів розкладу про помилки, що дозволить набагато швидше відновити актуальність (табл. 11).

Таблиця 10 – Сценарій «Збереження розкладу за посиланням»

Користувач	Вимоги
Викладач, студент	Кожен розклад повинен мати унікальну адресу, яку можна зберегти. При переході за цим посиланням повинен одразу відображатись збережений розклад, без необхідності виконання додаткових дій.

Таблиця 11 – Сценарій «Повідомлення про помилки у розкладі»

Користувач	Вимоги
Викладач, студент	Система повинна мати можливість надати повідомлення про помилки у розкладі. Таким чином модератори зможуть якнайшвидше виправити невідповідності.

2.2. Розробка архітектури

2.2.1. Проєктування моделей

Переважна більшість великих програмних систем використовують інформаційні бази даних. База даних може існувати як незалежно від програмної системи, так і може бути спеціально створена для розробленої системи. Важливою частиною моделювання систем є визначення логічної форми даних, що обробляються системою. Найбільш широко використовувана методологія моделювання даних – це моделювання типу «сутність-зв'язок-атрибут», що відображає структуру даних, їх атрибути та відносини між ними.

Сутність (entity) – реальний або абстрактний об'єкт, що має визначне значення для розглянутої системи (це може бути об'єкт як самої системи, так і його оточення). Кожна сутність повинна мати унікальну назву та мати один або декілька атрибутів, які або належать сутності, або наслідуються через зв'язок. Сутність відповідає класу (або типу) об'єктів, а не конкретному екземпляру класу.

Відношення (relation) – названий зв'язок (асоціація) між двома сутностями. Іменування зв’язків здійснюється за допомогою дієслів (наприклад, має, визначає, належить тощо).

Атрибут (attribute) – будь-яка характеристика сутності (призначена для ідентифікації, класифікації, численної параметризації або опису стану сутності). Значення атрибутів однозначно ідентифікують екземпляр сутності.

Для побудови діаграми моделі типу «сутність-зв'язок-атрибут» встановимо необхідні сутності з предметної області та опишемо їх атрибути. За базову візьмемо сутність «Університет» та визначимо її атрибути (табл. 12).

Таблиця 12 – Сутність «Університет»

Атрибут	Опис
id	Унікальний ідентифікатор університету
name	Повна назва
short_name	Скорочена назва
img	Символіка університету

У свою чергу до університету відносяться факультети (табл. 13) на яких містяться групи (табл. 14). Викладачі також відносяться до університету (табл. 15).

Таблиця 13 – Сутність «Факультет»

Атрибут	Опис
id	Унікальний ідентифікатор факультету
name	Повна назва
short_name	Скорочена назва

img	Символіка факультету
univ	Посилання на сутність «Університет»

Таблиця 14 – Сутність «Група»

Атрибут	Опис
id	Унікальний ідентифікатор групи
name	Назва
faculty	Посилання на сутність «Факультет»

Таблиця 15 – Сутність «Викладач»

Атрибут	Опис
id	Унікальний ідентифікатор викладача
full_name	ПІП
short_name	Короткий запис ПІП
degree	Ступінь
img	Фото
univ	Посилання на сутність «Університет»

Навчання в університеті поділяється на навчальні роки (табл. 16), а вони поділяються на семестри (табл. 17). Наостанок навчання представляється у вигляді пар, що викладаються на певних факультетах (табл. 18). Після опису всіх сутностей було створено діаграму моделі «сутність-зв'язок-атрибут»[17], яку наведено у додатку А.

Таблиця 16 – Сутність «Навчальний рік»

Атрибут	Опис
id	Унікальний ідентифікатор навчального року
start	Дата початку
end	Дата кінця
univ	Посилання на сутність «Університет»

Таблиця 17 – Сутність «Семестр»

Атрибут	Опис
id	Унікальний ідентифікатор семестру
num	Номер семестру
start	Дата початку
end	Дата кінця
year	Посилання на сутність «Навчальний рік»

Таблиця 18 – Сутність «Пара»

Атрибут	Опис
id	Унікальний ідентифікатор пари
name	Назва
housing	Місце проведення
room	Аудиторія
format	Тип пари
day	День тижня
subgroup	Номер підгруппи
teachers	Посилання на сутність «Викладач»
semester	Посилання на сутність «Семестр»

2.2.2. Проєктування інтерфейсу за допомогою дерева компонентів

Компонент є чистою функцією, тому його структура дуже проста. На вхід до компонента подається параметри з даними, а на виході ми отримуємо відрендерену частину інтерфейсу. Таким чином компонент при однаковості даних буде повертати однаковий рендер інтерфейсу. Компоненти можна компонувати для отримання складних інтерфейсів. Весь інтерфейс при цьому буде являтися чистою функцією від вхідних даних.

Під час проєктування інтерфейсу було складено діаграму (додаток Б), що описує весь інтерфейс, як композицію компонентів. З діаграми видно, що весь інтерфейс поділяється на піддерева, яким відповідають сторінки веб-інтерфейсу.

2.2.3. Побудова сервісу взаємодії з API-сервером

Взаємодія з сервером відбувається за рахунок розробленого сервісу, що включає набір класів для взаємодії с кожною кінцевою точкою API-серверу. Це надає системі низької зв’язності, що покращує підтримку. Інакше кажучи, при зміні способу взаємодії з сервером потрібно буде змінити лише цей сервіс, лишаючи інші частини додатку у незмінному вигляді.

Основою сервісу виступає базовий клас, що реалізує взаємодію на прикладному рівні із сервером. Таким чином інкапсулюючи однакову поведінку для кожної з кінцевих точок. Наприклад структуру запиту, або однакові заголовки. Об’єктна діаграма даного сервісу наведена у додатку В.

РОЗДІЛ 3. ВИЗНАЧЕННЯ РЕКОМЕНДАЦІЙ ЩОДО ОПТИМІЗАЦІЇ

3.1. Визначення параметрів оптимізації

Оптимізація додатку базується на збільшенні швидкості завантаження та роботи додатку. Для оцінки швидкодії важко визначити один універсальний параметр, тому доцільно проводити оцінку за рядом показників:

- час завантаження додатку;
- час завантаження побічних файлів та контенту;
- час до першого відображення інтерфейсу;
- швидкість обробки користувацьких дій.

Важливим щодо оптимізації є пошук потрібної точки в життєвому циклі розробки, щоб зробити це. Дональд Кнут влучно зауважив: "передчасна оптимізація є корінням всього зла" [18]. Сенс цих слів досить простий: досить легко втратити час, щоб отримати останній відсоток продуктивності в місцях, де це не матиме значного впливу. У той же час, деякі оптимізації заважають читабельності або придатності до підтримки, або навіть вносять помилки. Іншими словами, оптимізація повинна розглядатися не як "засіб для досягнення найкращої продуктивності програми", а як "пошук правильного способу оптимізації додатка та отримання найбільших переваг". Іншими словами, оптимізація всліпу може призвести до втрати продуктивності та прибутків.

Для дослідження додатків на предмет місць для можливих оптимізацій використовуються спеціальні інструменти – профайлери. Вони дозволяють досліджувати внутрішні процеси роботи браузера, що використовуються для роботи додатку та знаходити проблемні місця, які варто оптимізувати, не погіршуючи розробку та підтримку.

Шкала часу в інструментах розробника Chrome (рис. 5) може допомогти знайти операції, що виконуються надто довго.

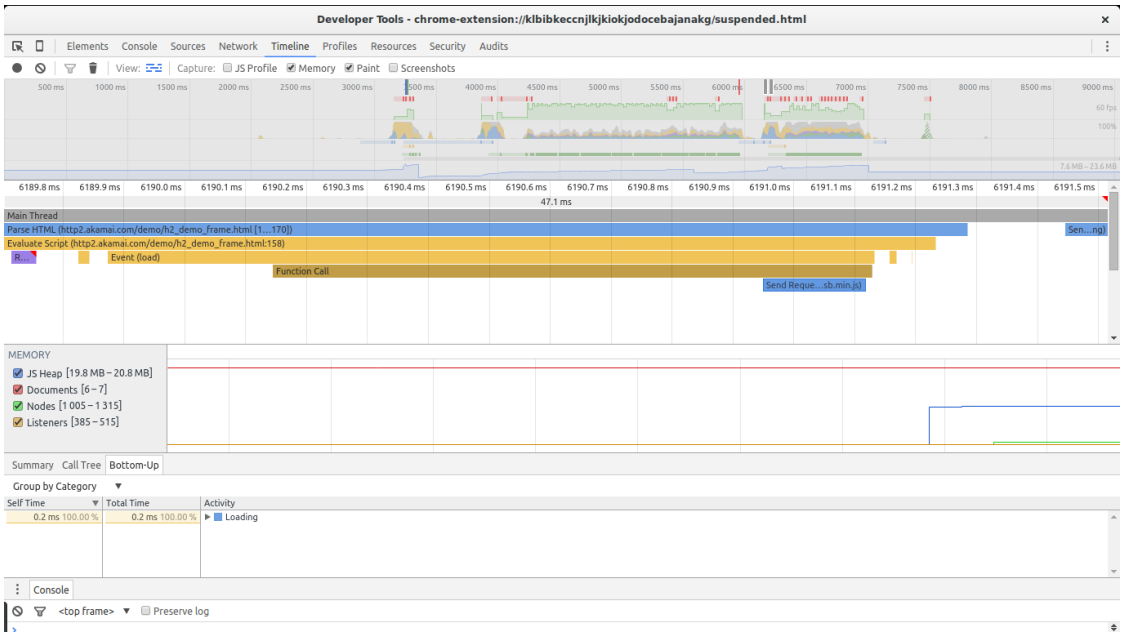


Рис. 5 – Профайлер шкали часу в інструментах розробника Chrome

Перегляд мережевих взаємодій (рис. 6) дозволяє визначити додаткову затримку, що може з’явитися через завантаження надто важких файлів, або повільних запитів до серверу.

Використання пам'яті – це ще одна характеристика, яка може призвести до успіху, якщо буде правильно проаналізована. Якщо розроблена сторінка містить багато візуальних елементів (великі, динамічні таблиці) або багато інтерактивних елементів (наприклад, ігри), оптимізація пам'яті може призвести до меншого гальмування додатку та збільшення частоти кадрів.

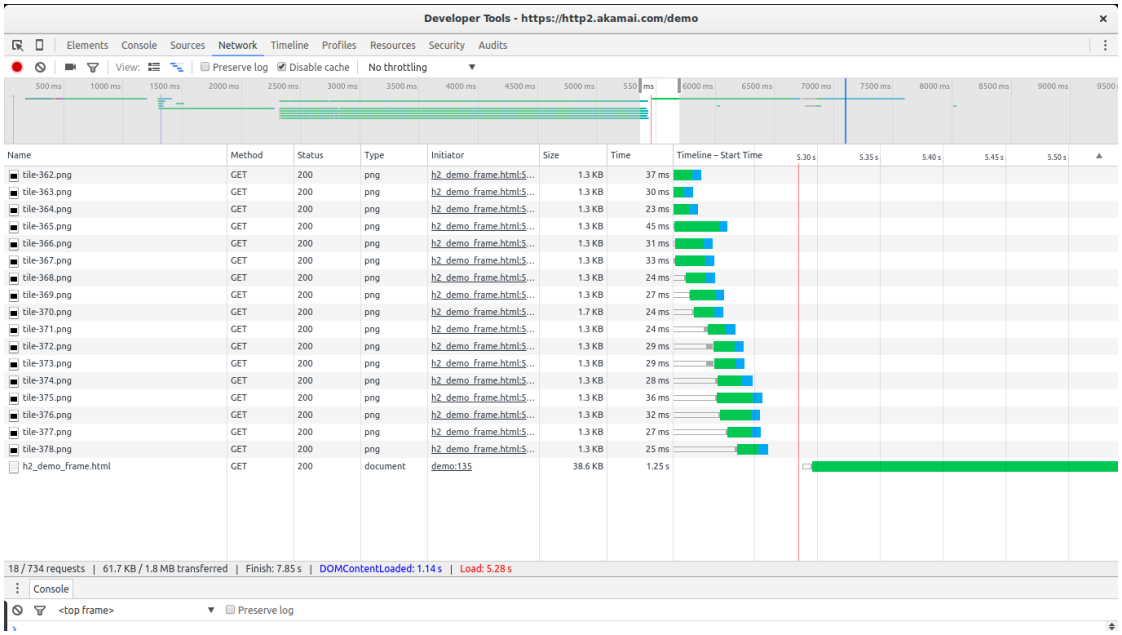


Рис. 6 – Профайлер мережевих запитів в інструментах розробника Chrome

Відмалювання – це процес зафарбування пікселів. Це часто найтриваліша частина процесу відображення. Якщо додаток працює неприйнятно, то скоріше за все це вказує на проблеми з відмалюванням

Композиция – процес, що об'єднує відмальовані частини сторінки для відображення на екрані. Якщо оптимізувати додаток на використання у більшості випадків композиції і зовсім уникати відмальовування, то можна помітити значний приріст працездатності. Дослідити процес відмалювання можна у профайлері відмальовки (рис. 7)

Класифікація методів оптимізації:

– за рахунок зменшення розміру завантажуваних файлів;
– оптимізація роботи коду додатку;
– оптимізація рендерингу;
– інші методи покращення швидкодії.

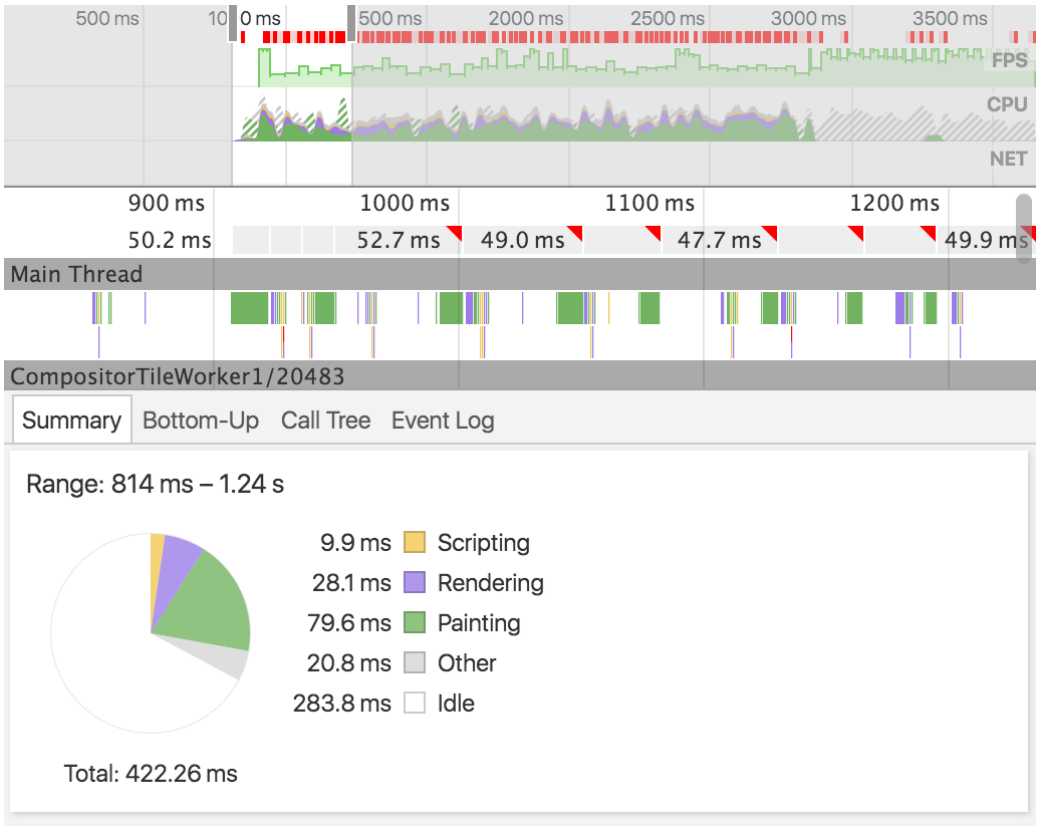


Рис. 7 – Профайлер відмалювання в інструментах розробника Chrome

3.2. Зменшення розміру виконуваного файлу

JavaScript-додатки розповсюджуються у вигляді сирцевого коду. Розбір такого коду є менш ефективним, ніж аналіз байт-коду нативних додатків. Для невеликих сценаріїв різниця незначна, однак для більших додатків розмір сценарію може негативно впливати на час запуску додатка. Внесок цієї проблеми можна редукувати за рахунок зменшення розміру коду – мініфікації. Це водночас є найбільш доступним і дієвим засобом зменшення розміру додатку. Даний процес прибирає надлишкові символи – пробіли та табуляцію без шкоди працездатності та функціональності. Окрім цього мініфікація може прибирати змінні та частки коду, що не виконуються.

3.2.1. Видалення надлишкового коду

Для процесу видалення надлишкового (мертвого) коду використовується так званий метод Tree Shaking. Цей термін зазвичай використовується лише в контексті JavaScript. Метод спирається на дослідження об’явлених імпортованих та експортованих бібліотек в додатку, щоб виявити, використовується імпортований код, чи ні. У сучасних застосунках JavaScript використовуються інструменти для автоматичного видалення мертвого коду при об'єднанні декількох файлів JavaScript в один файл – Webpack, Rollup.

Перевіримо даний вид оптимізації на нашому додатку. Побудуємо мапу залежностей (рис. 8) додатку та проаналізуємо його розмір до оптимізації та після.

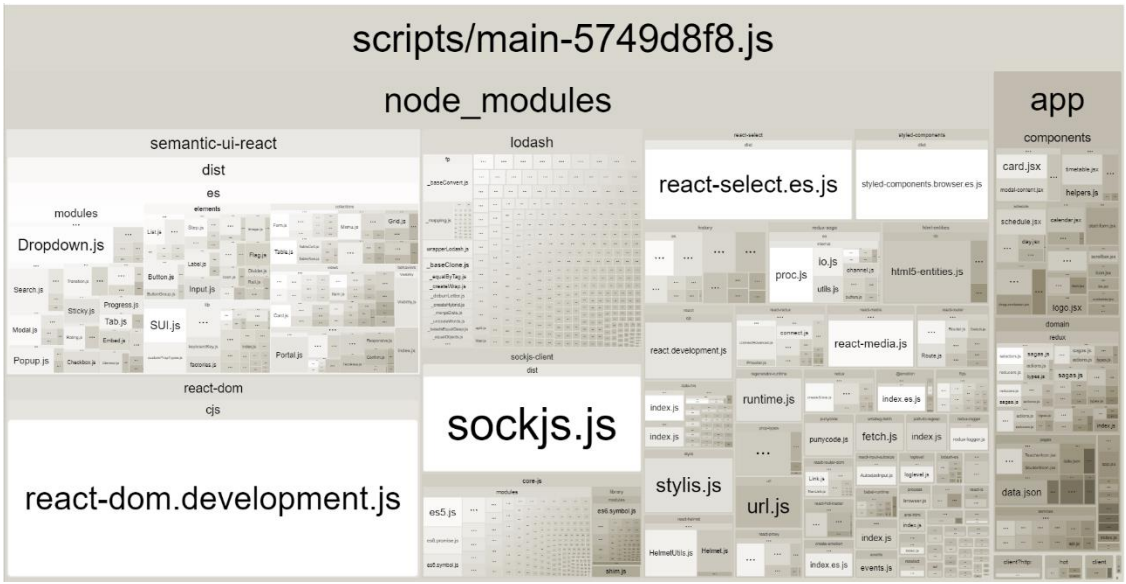


Рис. 8 – Мапа залежностей додатку без оптимізацій

Розмір додатку до мініфікації та видалення надлишкового коду становить 4.24 Мб. Після проведення процесу оптимізації розміру за рахунок видалення зайвих надлишкових бібліотек та мініфікації коду розмір додатку

становить 1.07 Мб, що значить, що оптимізація допомогла зменшити розмір додатку вчетверо.

Подивимось на діаграму залежностей оптимізованого додатку (рис. 9). Чітко видно, що на відміну від мапи до оптимізації (див. рис. 8) на цій відсутні великі бібліотеки react-dom.development.js та react-select.es.js. Дійсно, якщо перевірити розмір цих бібліотек, в сумі отримаємо 1.4 Мб. Тоді можна зробити висновок, що останні 2.8 Мб були видалені за рахунок мініфікації та стискання коду.

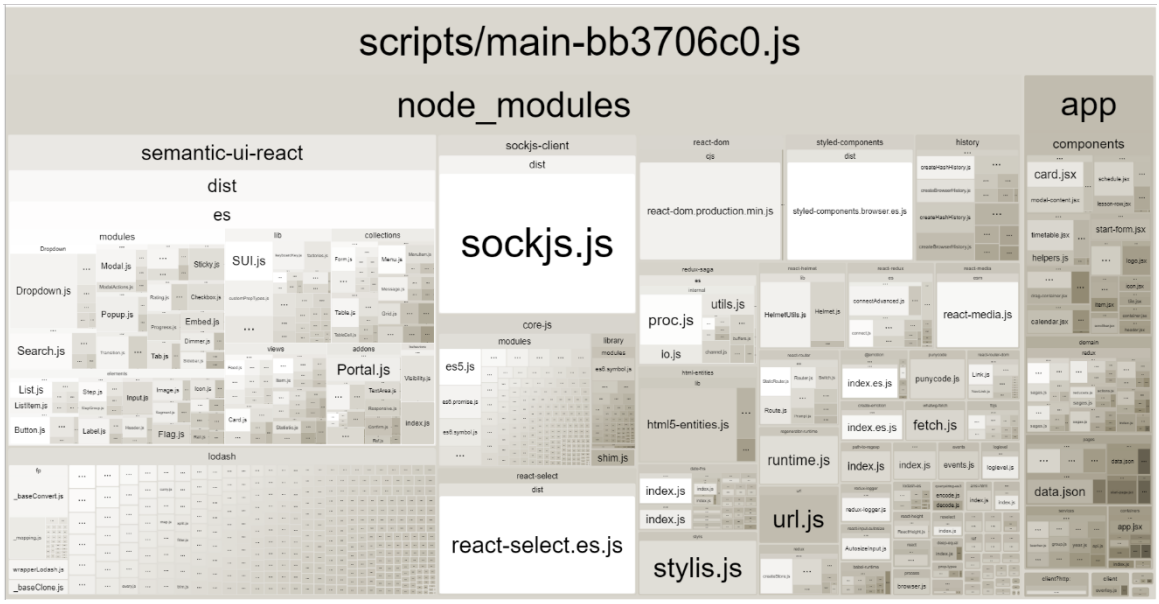


Рис. 9 – Мапа залежностей додатку після оптимізації розміру

3.2.2. Розділення коду та ліниве підвантаження

Видалення мертвого коду зачіпає лише код, який зовсім не використовується. Але по мірі росту додатку з’являється функціонал, що використовується дуже рідко, але код мертвим не є. Особливо це помітно, коли такий функціонал використовує важкі бібліотеки для своєї роботи. Було б

оптимально позбавити основний файл додатку такого функціоналу та виділити його в окремі файли, що можуть підвантажуватися за потребою.

Таке розділення коду на декілька файлів з підвантаженням за потребою називається Code Splitting (розділення коду). Ця функція підтримується основними сбірниками коду, як Webpack і Browserify, що створюють декілька пакетів, які можуть динамічно завантажуватись під час роботи програми, якщо виникне потреба у функціоналі з цих файлів – це називається Lazy Loading (ліниве підвантаження).

Принцип лінивого підвантаження гарантує, що буде завантажуватись лише той код, який потрібен користувачеві у даний момент для роботи з додатком, таким чином підвищуючи продуктивність додатку. Хоча це і не впливає на загальний розмір коду, але допомагає зробити початкове завантаження додатку швидшим.

Така методика дозволила зменшити розмір початкового файлу ще приблизно на 530 Кб. Результуючий файл тепер важить 482 Кб – в два рази менше. А увесь додаток тепер розділений на 5-8 файлів, що можуть завантажуватись за потребою у певному функціоналі.

3.3. Зменшення розміру зображень

Зображення – ті ресурси, які часто займають багато місця на сторінці і важать найбільше. Завдяки їх оптимізації ми можемо значно зменшити кількість завантажуваних даних і поліпшити роботу сайту. Чим більше стиснене зображення, тим менше пропускної здатності каналу займає скачування і тим швидше браузер зможе показати сторінку користувачеві.

Оптимізація зображень – це одночасно наука і мистецтво. Ми можемо назвати це мистецтвом, тому що ніхто не може дати певну відповідь, як найкраще стиснути конкретне зображення. Однак це і наука, адже в нашому

розпорядженні є розроблені техніки і алгоритми, які можуть значно зменшити розмір ресурсу. Щоб вибрати оптимальні налаштування для зображення, необхідно врахувати багато чинників: можливості формату, закодовані дані, якість, кількість пікселів тощо.

Для початку необхідно вибрати відповідний формат для зображення:

- у векторній графіці для відображення картинки використовуються лінії, точки і багатокутники;
- у растровій графіці кодуються індивідуальні значення кожного пікселя в прямокутній сітці, і на їх основі будується зображення.

У кожного формату є свої переваги і недоліки. Векторний формат ідеально підходить для зображень з простих геометричних фігур (наприклад, логотипів, тексту, значків тощо). Вони залишаються чіткими при будь-якій роздільній здатності і масштабі, тому слід використовувати цей формат для великих екранів і ресурсів, які потребують відображення в різних розмірах.

Однак векторні формати не підходять для складних зображень (наприклад, для фотографій). SVG-коду для опису складних форм може стати занадто багато, але отримане зображення все одно буде виглядати нереалістично. В цьому випадку вам варто використовувати растровий формат зображень, наприклад GIF, PNG, JPEG або нові формати JPEG-XR і WebP.

Якість растрових зображень залежить від роздільної здатності і масштабу: при збільшенні воно стає розмитим і розпадається на пікселі. В результаті може знадобитися зберегти кілька версій растрового зображення в різних роздільних здатностях.

Один з найпростіших способів оптимізації зображення – знизити глибину кольору з 8 бітів на канал, вибравши палітру меншого розміру. Встановивши глибину в 8 бітів на канал, ми отримуємо 256 значень для каналу і 16 млн. кольорів. Якщо зменшити палітру до 256 кольорів, нам буде потрібно

всього 8 біт для всіх каналів RGB і тільки 2 байти на піксель, а не 4, як раніше. Таким чином розмір зображення зменшиться в два рази.

В останні роки найкращі показники стиснення зображень має формат WebP, що може зменшити розмір зображення на 30% порівняно з форматами PNG та JPEG з такою ж візуальною якістю [19].

Спираючись на інформацію про формати зображень та їх можливості виділимо алгоритм для підбору оптимального типу зображення:

- якщо зображення повинно бути анімованим, то потрібно однозначно обирати формат GIF. Колірна палітра GIF складається всього з 256 кольорів. Це недостатньо для більшості зображень. Крім того, формат PNG-8 краще стискає зображення з маленькою палітрою. Таким чином, потрібно обирати GIF, лише якщо вам потрібно анімоване зображення;
- деякою альтернативою анімованому зображенню є відео, але такий спосіб складний у реалізіції. Хоча відео має набагато краще стиснення та більше можливостей у контролі програвання;
- якщо потрібно зберегти всі дрібні деталі в найвищій якості – потрібно використовувати PNG. У форматі PNG не застосовується стиснення з втратою даних, не рахуючи вибору розміру палітри. Завдяки цьому зображення зберігається в найвищій якості, але важить набагато більше, ніж файли інших форматів. Використання цього формату є доцільним лише збереження деталізації або для картинок з прозорістю;
- якщо зображення складається з геометричних фігур то кращим рішенням буде конвертація його в векторний (SVG) формат;
- якщо потрібно оптимізувати фотографію, скріншот або подібні зображення – слід використати формат JPEG. В JPEG

використовується комбінація стиснення з втратами і без втрат для зменшення розміру файлу. Щоб вибрати найкраще поєднання якості і розміру зображення, необхідно окремо для кожного файлу встановити баланс між рівнем якості JPEG і візуальною якістю зображення для забезпечення мінімального розміру.

3.4. Кешування

Кеш-пам'ять – це місце, де зберігаються статичні дані, до яких часто звертається додаток, так що подальші запити на ці дані можуть обслуговуватися швидше або ефективніше. Оскільки веб-програми вирізняються дуже мінливим інтерфейсом, кеш можна знайти використовним в багатьох частинах архітектури додатку. Наприклад, кеш-пам'ять може бути встановлена між динамічним контент-сервером та клієнтами, щоб запобігти збільшення завантаження сервера запитами до популярних ресурсів і одночасно поліпшити час відгуку. Інші кеші можуть бути поміщені перед базами даних або процесами, що вимагають багато ресурсів. Тобто, кеш-пам'ять – чудовий спосіб поліпшити час відгуку та зменшити використання процесора в веб-додатках.

Нещодавно в браузерах з’явилася можливість використовувати так званий Service Worker – скрипт, який працює у фоновому режимі, окремо від веб-сторінки, відкриваючи можливості користатися функціями, які не потребують веб-сторінки або взаємодії з користувачем. Основна особливість, характерна для них – можливість перехоплювати та обробляти запити мережі, включаючи програмне керування кешем відповідей сервера. Цей функціонал дозволяє розробляти додатки, що набагато менше залежать від мережі та навіть можуть працювати оффлайн.

Технологія Service Worker дозволяє закешувати будь-який файл для сайту і при оновленні цього файлу підмінити його. Це дає приріст у роботі додатків, що часто використовуються. Така система успішно використовується в поштових клієнтах, клієнтах соціальних мереж, та невеликих веб-додатках, що зовсім не потребують роботи з мережею.

3.5. Оптимізація рендерингу

Для того щоб писати продуктивні сайти і додатки, необхідно розуміти, яким чином браузер обробляє HTML, JavaScript і CSS. А вже на основі цих знань необхідно докласти зусиль, щоб код (в тому числі і сторонній код) працював якомога ефективніше.

Сьогодні більшість пристроїв оновлюють свої екрани 60 раз в секунду. Якщо виконується анімація або перехід або якщо користувач прокручує сторінку, браузеру потрібно відповідати частоті оновлення екрану пристрою і видавати по одному новому кадру при кожному оновленні екрана.

Кожен з цих кадрів може тривати трохи більше 16 мс (1 секунда / 60 = 16,66 мс). В реальності ж браузеру потрібно виконати і ще деякі дії, тому робота коду повинна займати не більше 10 мс. Якщо не вкластися в ці рамки, частота кадрів буде менше, а контент почне смикатися на екрані. Часто цю ситуацію називають підвисанням, вона негативно впливає на сприйняття користувачів.

Є п'ять основних областей що стосуються рендерингу, про які слід знати і пам'ятати при написанні веб-додатку. Це ті області, які піддаються найбільшому контролю. Вони є ключовими точками конвеєра виведення пікселів на екран.

- JavaScript – зазвичай JavaScript використовується для виконання роботи, результатом якої будуть візуальні зміни. Але такі зміни можна викликати і за рахунок CSS, інтерфейсу анімації тощо;
- обчислення стилів – в процесі обчислення визначається, які правила на які елементи будуть впливати. Після того, як правила визначені, вони застосовуються для кожного елемента;
- обрахунок макету – як тільки браузер дізнається, які правила застосовуються до елемента, він може почати вираховувати його положення та розмір на екрані. Модель макету для сайту означає, що один елемент може впливати на інший через набір властивостей, тому ці підрахунки можуть бути дуже важкими для браузера;
- відмальовування – це процес заповнення пікселів. Він передбачає вивід тексту, кольорів, зображень, границь, тіней – усіх візуальних частин інтерфейсу. Відмальовування зазвичай відбувається на декількох поверхнях, які називаються шарами;
- компонування – оскільки частини сторінок потенційно були відмальовані на декількох шарах, вони повинні бути виведені на екран в правильному порядку, щоб сторінка відображалася правильно. Це особливо важливо для елементів, що перекривають інші елементи, оскільки помилка може призвести до неправильного відображення нижнього елемента на поверхні першого.

Як бачимо, рендеринг – дуже складний і многосторонній процес, тому необхідно якомога рідше викликати такі зміни у інтерфейсі, що можуть вплинути на швидкість та якість відображуваного інтерфейсу. Одна з найбільш поширених проблем – виклик рендерингу кілька разів для однієї зміни інтерфейсу. Цю проблему може вирішити тактика роботи з інтерфейсом, коли всі необхідні обчислення робляться у максимальній кількості до

рендерингу і тільки потім застосовуються до інтерфейсу. Найбільш досконалою реалізацією такої поведінки можна вважати Virtual DOM. Він виконує обчислення інтерфейсу і застосовує зміни тільки у разі надходження змінених даних, інакше інтерфейс не перемальовується.

Однак мутація змінних під час роботи додатку передбачає, що ми повинні порівняти всю структуру даних на предмет зміни. Це є дуже незручний та довготривалий процес. Тому використовується поняття імутабельності, що значить, що зміна даних відбувається за допомогою їх копіювання, а старі структури залишаються недоторканими. Тому перевірити зміну даних можна провівши порівняння об’єктів (їх адрес). Розбіжність у них буде казати про зміну, навіть якщо вона відбулася глибоко всередині структури даних.

В загальному випадку Virtual DOM дозволяє проводити рендеринг лише компонентів зі зміненими даними. А імутабельність даних зменшує кількість затраченого часу на повне порівняння минулого стану з теперішнім, дозволяючи порівнювати лише адреси на об’єкти з даними.

3.6. Результати оптимізації додатку

Методи оптимізації, зазначені в попередніх пунктах були випробувані на розробленому додатку. Для визначення якості оптимізації було виконано виміри за параметрами згаданими у п. 3.1 до оптимізації та після. Отримані значення були занесені у таблицю (табл. 19) для порівняння.

Оптимізації позитивно позначилися на працездатності додатку, помітне пришвидшення за деякими параметрами більше, ніж в два рази.

Таблиця 12 – Порівняння результатів оптимізації веб-інтерфейсу

	До оптимізації	Після оптимізації
час завантаження додатку (с)	10	3
час завантаження побічних файлів та контенту (с)	6	5
час до першого відображення інтерфейсу	20	10
швидкість обробки користувацьких дій (кадрів/с)	~40	~50-60

ВИСНОВКИ

У результаті виконання роботи було досліджено засоби розробки веб-додатків, а саме інтерфейсів для них. Дослідження включало такі засоби, як архітектурні патерни, бібліотеки та фреймворки для роботи з UI, бібліотеки обробки даних, бібліотеки роботи зі станом додатку та бібліотеки тестування. Були описані основні способи їх застосування, особливості роботи та основні переваги, недоліки. Найкращими у своїх принципах роботи, зручності та опрацьованості практик використання виявились бібліотеки React, тому їх було вирішено використати для розробки додатку.

Спроєктовано сценарії роботи, пророблено сутності та побудовано компонентну діаграму додатку. Це допомогло досягнути визначеного у вимогах функціоналу без виникнення складнощів і нерозв’язних проблем у визначений час. Інтерфейс, що пропонує додаток, дозволяє швидко знайти потрібний розклад за університетом, факультетом та групою. Знайдений розклад можна зберегти у вигляді посилання і мати доступ до цієї інформації без повторного пошуку. Розроблений додаток є універсальним, а отже відмінно працює на будь-яких пристроях: як на телефонах, так і на комп’ютерах.

Досліджено практики оптимізації швидкості роботи додатку та розроблено рекомендації щодо них. Їх було протестовано на розробленому додатку і показали відмінний результат, збільшивши швидкість завантаження, роботи та покращивши суб’єктивне враження від інтерфейсу. Серед практик було розглянуто мінімізацію сирцевого коду, видалення мертвого коду, практики вибору оптимального формату зображення та зменшення його розміру, оптимізацію процесу рендерингу та кешування. Розроблений додаток

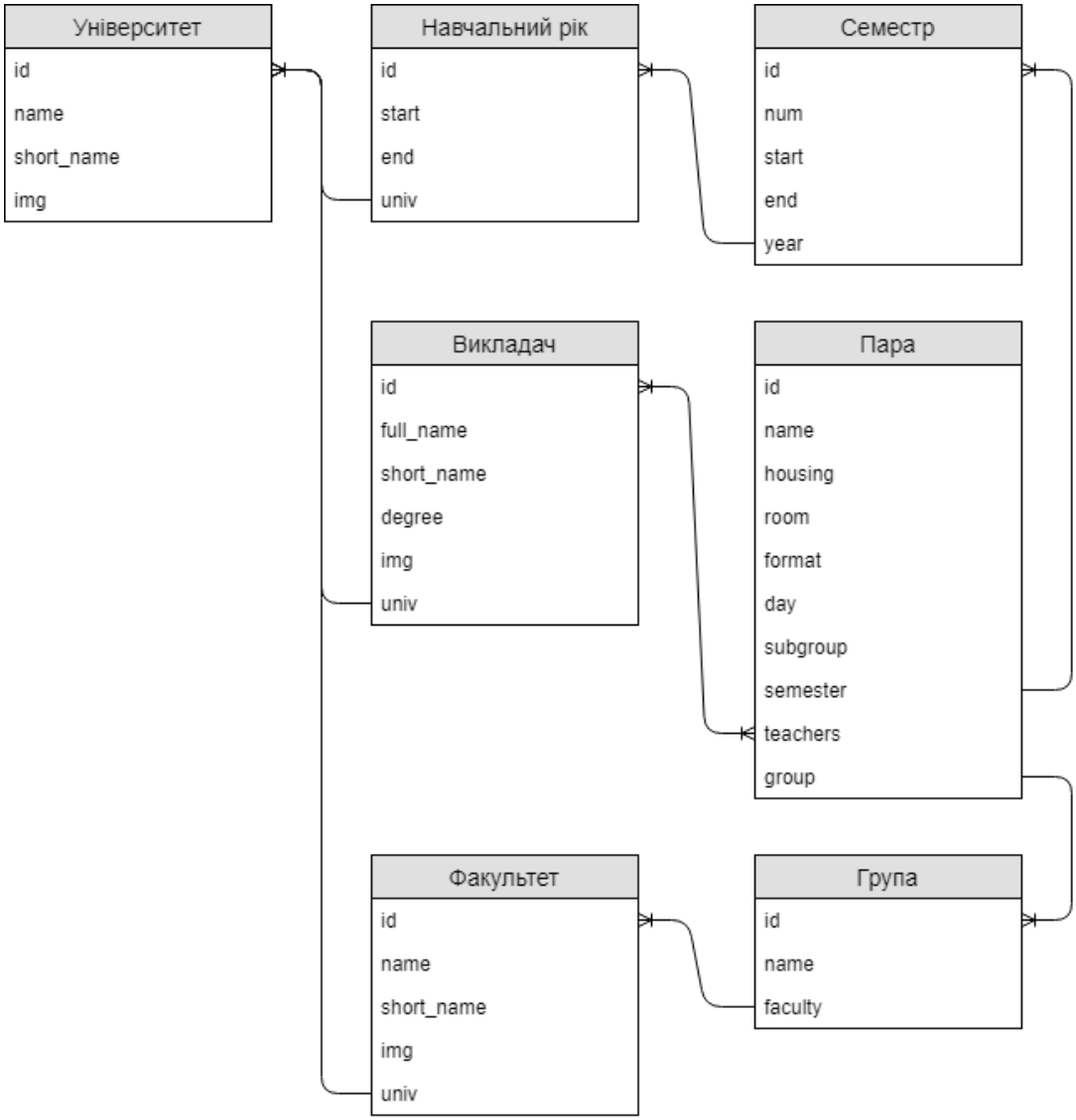
має велику практичну значущість, так як допомагає у навчальному процесі викладачам і студентам, забезпечуючи зручний доступ до розкладу.

ПЕРЕЛІК ДЖЕРЕЛ ТА ПОСИЛАННЯ

1. Renskaug T., Coplien J. The DCI Architecture: A New Vision of Object-Oriented Programming – March 20, 2009. – Ел. ресурс. Режим доступу: http://www.artima.com/articles/dci_vision.html
2. Burbeck S. Applications Programming in Smalltalk-80: How to use Model–View–Controller (MVC). – 09.2012. – Ел. ресурс. Режим доступу: http://www.dgp.toronto.edu/~dwigdor/teaching/csc2524/2012_F/papers/mvc.pdf.
3. Simple Example of MVC (Model–View–Controller) Design Pattern for Abstraction. – Ел. ресурс. Режим доступу: http://www.codeproject.com/Articles/25057/Simple-Example-of-MVC-Model-View-Controller-Design.
4. Schmidt D. Pattern-Oriented Software Architecture / D. Schmidt, M. Stal, H. Rohnert, F. Buschmann. – C: Wiley, 1996. – 476 с.
5. MVC architecture: The theory behind Model View Controller. – 2018. – Ел. ресурс. Режим доступу: https://developer.mozilla.org/en-US/Apps/Fundamentals/Modern_web_app_architecture/MVC_architecture.
6. GWT Project. – Ел. ресурс. Режим доступу: https://developers.google.com/web-toolkit/articles/testing_methodologies_using_gwt.
7. Facebook Flux: In Depth Overview. – Ел. ресурс. Режим доступу: https://facebook.github.io/flux/docs/in-depth-overview.html.
8. GitHub: angular.js. – Ел. ресурс. Режим доступу: https://github.com/angular/angular.js.
9. VueJS. Simplified JavaScript Jargon. – Ел. ресурс. Режим доступу: http://jargon.js.org/_glossary/VUEJS.md.

10. Krill P. React: Making faster, smoother UIs for data-driven Web apps / P. Krill. – 2014. – Ел. ресурс. Режим доступу: https://www.infoworld.com/article/2608181/javascript/react--making-faster--smoother-uis-for-data-driven-web-apps.html.
11. Hemel Z. Facebook's React JavaScript User Interfaces Library Receives Mixed Reviews / Z. Hemel. – 06. 2013. – Ел. ресурс. Режим доступу: https://www.infoq.com/news/2013/06/facebook-react
12. Dawson Ch. JavaScript’s History and How it Led To ReactJS / Ch. Dawson. – 2014. – Ел. ресурс. Режим доступу: https://thenewstack.io/javascripts-history-and-how-it-led-to-reactjs/.
13. Murali S. Slideshare: Yahoo Mail moving to React / S. Murali. – 2014. – Ел. ресурс. Режим доступу: https://www.slideshare.net/rmsguhan/react-meetup-mailonreact.
14. Wren Ch. Dev Chats: Spike Brehm of Airbnb / Ch. Wren. – 2014. – Ел. ресурс. Режим доступу: https://medium.com/code-stories/dev-chats-spike-brehm-of-airbnb-87e155f3475d.
15. MobX: CoreConcepts. – Ел. ресурс. Режим доступу: https://mobx.js.org/index.html.
16. Соммервилл И. Инженерия программного обеспечения, 6-е издание / И. Соммервилл. – М. : Издательский дом "Вильямс", 2002. – 624 с.
17. Chen P. The entity relationship model - Towards a unified view of data / P. Chen. – M. : ACM Trans, on Database Systems, – 1976. – c. 9-45.
18. Knuth D. E. Structured Programming with go to Statements // Computing Surveys. – 1974. – V. 6. – No. 4. – December 1974. Stanford University, Stanford, California. – 262 p.
19. WebP: Compression Techniques. – 2016. – Ел. ресурс. Режим доступу: https://developers.google.com/speed/webp/docs/compression.

ДОДАТОК А
ДІАГРАМА «СУТНІСТЬ-ЗВ'ЯЗОК-АТРИБУТ» РОЗРОБЛЮВАНОГО ВЕБ-ІНТЕРФЕЙСУ



ДОДАТОК Б
ДІАГРАМА КОМПОНЕНТНОЇ СТРУКТУРИ ВЕБ-ІНТЕРФЕЙСУ

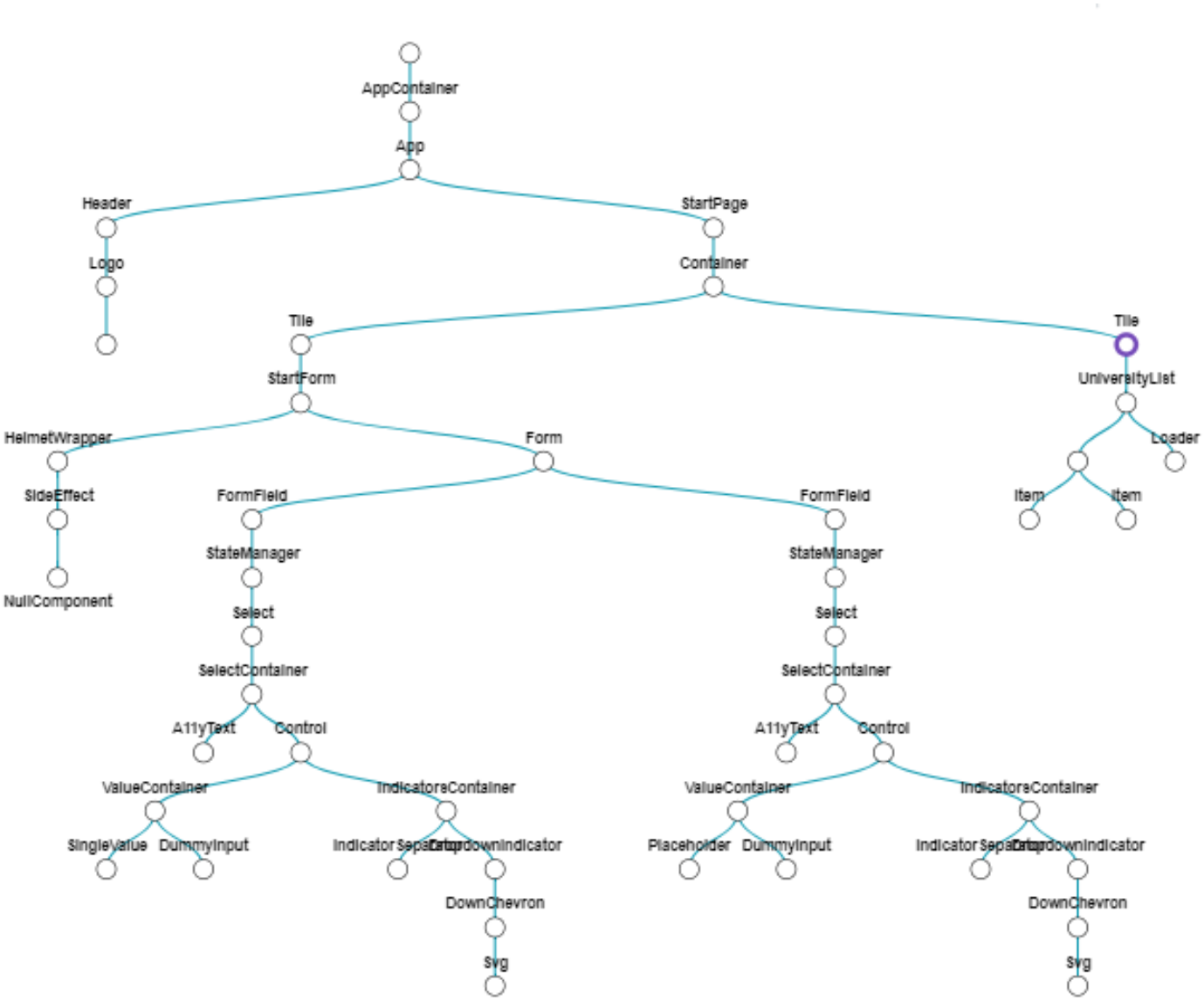


Рис. Б.1 – Компонентна діаграма сторінки пошуку розкладу

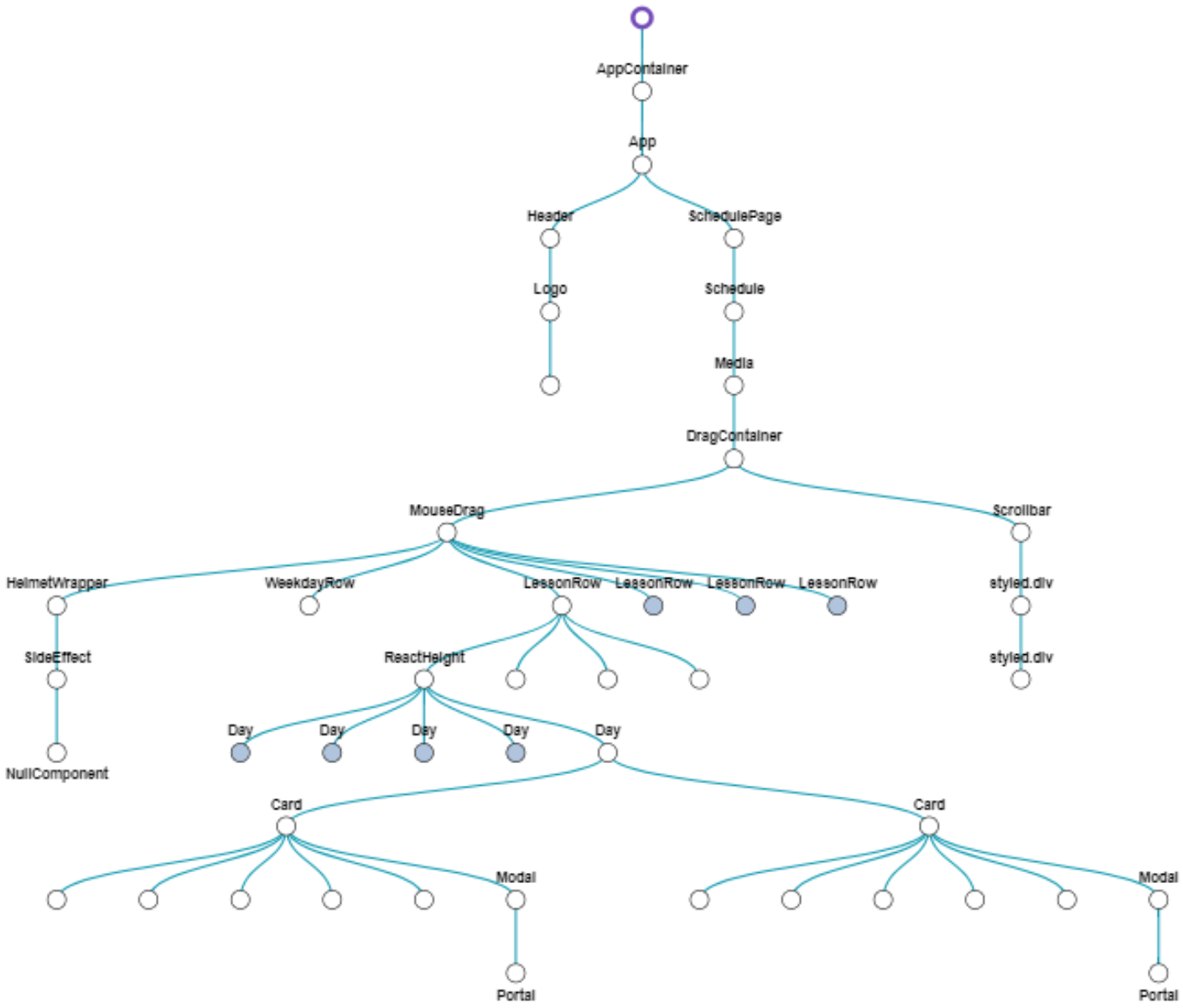


Рис. Б.2 – Компонентна діаграма сторінки розкладу групи

ДОДАТОК В
ОБ’ЄКТНА ДІАГРАМА API-СЕРВІСУ