Міністерство освіти і науки України

Національний університет «Києво-Могилянська Академія»

Кафедра мультимедійних систем

КУРСОВА РОБОТА

за спеціальністю «Інженерія програмного забезпечення» 121

на тему:

Дослідження можливостей фреймворку Vue.js на прикладі розробки веб-застосунку для контролю власних фінансів

Науковий керівник:

ст. викладач Вовк Н. Є.

Виконав:

студент 3-го курсу Скрипнік А. О.

Київ – 2020

Міністерство освіти і науки України

Національний університет «Києво-Могилянська Академія»

Кафедра мультимедійних систем

ЗАТВЕРДЖУЮ

Зав. кафедри мультимедійних систем,

доцент, к.ф.-м.н.

_____О. П. Жежерун

“____”_____ 2020 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на курсову роботу

студенту Скрипніку Андрію 3-го курсу факультету інформатики

ТЕМА: Дослідження можливостей фреймворку Vue.js на прикладі розробки веб-застосунку для контролю власних фінансів

Зміст ТЧ до курсової роботи:

Анотація

Вступ

1. Огляд підходів розробки клієнтської частини веб-застосунку
2. Архітектура клієнтської частини веб-застосунку на Vue.js
3. Огляд додаткових можливостей на базі фреймворку Vue.js
4. Реалізація практичної частини роботи

Висновки

Список джерел

Додатки (за необхідністю)

Дата видачі “___” _____ 2020 р.

Керівник _____ Завдання отримано _____

Календарний план виконання курсової роботи

Тема: Дослідження можливостей фреймворку Vue.js на прикладі розробки веб-застосунку для контролю власних фінансів

№ п/п	Назва етапу курсового проекту (роботи)	Термін виконання етапу	Примітка
1.	Отримання завдання на курсову роботу	листопад 2019 р.	
2.	Огляд документації та літератури за темою роботи	листопад - грудень 2019 р.	
3.	Оволодіння навичками веб-розробки з використанням Vue.js	грудень - лютий 2020 р.	
4.	Реалізація практичної частини роботи	лютий - березень 2020 р.	
6.	Написання теоретичної частини курсової роботи	березень 2020 р.	
7.	Надання роботи керівнику для перевірки, демонстрація практики	квітень 2020 р.	
8.	Корегування роботи за результатами перевірки керівником	квітень 2020 р.	
9.	Остаточне оформлення теоретичної частини та слайдів доповіді	квітень-травень 2020 р.	
10.	Захист курсової роботи	травень 2020 р.	

Студент _____ Скрипнік А.О. _____

_____

Керівник _____ Вовк Н.Є. _____

_____

“_____” _____ _____ р.

Зміст

Анотація

Роботу присвячено вивченню й аналізу архітектурних та програмних рішень, наданих фреймворком Vue.js, для розробки клієнтської частини веб-застосунків.

Результат роботи відображено у веб-системі контролю власних фінансів, розробленої з використанням Vue.js та інших технологій, побудованих на базі фреймворку або безпосередньо пов’язаних із ним.

Ключові слова: Vue.js, Javascript, Vuetify, Vuex, Vue Router, REST API, Vue-resource, REST API, JSON, Kotlin, Spring Boot, Google OAuth2, Webpack, Gradle, PostgreSQL, single page application (SPA), реактивне програмування.

Вступ

Більшість сучасного середнього та великого бізнесу потребує веб-застосунок або мобільний додаток для маркетингової привабливості та конкурентоспроможності у своїй сфері діяльності. Навіть якщо це правило не виконується для деяких специфічних сфер, то внутрішня система для організації процесів виробництва є необхідною умовою для функціонування підприємства.

Очевидно, що інтуїтивна зрозумілість системи, її швидкість роботи та дизайн безпосередньо впливають на користувацький досвід людини-споживача системи. Якими б складними та великими за обсягом не були внутрішні функції системи – користувачу бажано мати інтерфейс, який би в найшвидші терміни задовольнив його потреби.

Звісно, існує багато рішень цього актуального питання. В сучасному світі ці рішення з’являються навіть занадто швидко, проте немає певної системності та однозначності при виборі однієї чи іншої технології.

Я обрав тему, пов’язану саме з фреймворком Vue.js, адже ця молода технологія, на мій погляд, є недооціненою в плані розуміння повноти обсягу її можливостей.

Моєю метою є аналіз, систематизація та опис аспектів розробки повноцінного веб-застосунку з використанням Vue.js та інших технологій, безпосередньо пов’язаних з цим фреймворком.

Методика досягнення мети – практика, що базується на значному теоретичному фундаменті.

Робота складається з чотирьох розділів.

Перший розділ присвячено огляду існуючих підходів розробки клієнтської частини веб-застосунків. Розглянуті найпопулярніші архітектурні рішення та інструменти їх реалізації.

Другий розділ присвячено дослідженню архітектури клієнтської частини веб-застосунку з використанням фреймворку Vue.js. Проаналізована реактивна парадигма. Досліджена система компонентів та особливості їх взаємодії. Коротко описана історія фреймворку.

Третій розділ присвячений додатковим можливостям розробки клієнтської частини на базі Vue.js. Наведені приклади бібліотек, які легко інтегруються з Vue з метою розширення функціональних можливостей веб-інтерфейсу.

Четвертий розділ присвячений опису реалізації веб-застосунку, клієнтська частина якого побудована на основі Vue.js з використанням рекомендованих практик розробки та великої кількості функціональних можливостей, наданих фреймворком. Проаналізована швидкість опанування технологіями, зручність використання фреймворку з боку розробника та кінцевий результат для користувача.

Практична частина курсової роботи відображена у готовому програмному продукті – веб-застосунок для контролю власних фінансів. Продукт розгорнутий на віддаленому сервері та доступний з будь-якого пристрою, підключеного до мережі Інтернет.

За результатами аналізу та дослідження можливостей фреймворку Vue.js зроблені відповідні висновки.

1 Огляд підходів розробки клієнтської частини веб-застосунку

1.1 Статичний HTML

Довгий час чи не єдиним підходом до розробки клієнтської частини веб-застосунку були статичні HTML сторінки, генеровані сервером. Тобто сама клієнтська частина – набір статичних сторінок, які змінюють одна одну при відповідному HTTP запиті віл клієнта.

Сервер генерує сторінку за допомогою шаблонізаторів, які відрізняються для кожної мови програмування. Вони є двигунами динамічної генерації HTML сторінки, які в залежності від даних, наданих сервером, інтерпретують умовні конструкції чи цикли у відповідну розмітку. Прикладами шаблонізаторів є: JSP (для Java), Twig (для PHP), EJS (для Node.js).

Засобами динамічних змін на клієнтській частині при такому архітектурному підході є клієнтський JavaScript, зокрема бібліотека jQuery, який підключається до відповідної сторінки всередині тегу < script >.

Перевагою такого підходу є простота в розробці, адже по суті потрібне знання HTML розмітки, CSS стилів та базового JavaScript для розробки повноцінної клієнтської частини.

Найбільшими недоліками є зав’язаність на серверну частину та необхідність при кожному запиті повністю перезавантажувати сторінку, що може мати негативний вплив на продуктивність та швидкість роботи системи.

На нинішній момент часу даний підхід є досі широко застосований в багатьох веб-застосунках, проте з кожним роком його позиції слабшають.

1.2 Односторінковий застосунок (single-page application, SPA)

Із розвитком веб-технологій набрали популярність так звані односторінкові застосунки. Вони складаються із однієї HTML сторінки-оболонки, в яку динамічно завантажується контент [1].

Таким чином сервер не виступає в ролі необхідної ланки для генерування представлення даних на клієнтській частині застосунку. Сервер є звичайним постачальником даних, що зберігаються в базі. Найчастіше односторінкові застосунки розробляються згідно з RESTful архітектурою: клієнт та сервер обмінюються даними в зручному форматі (наприклад, JSON) з використанням HTTP-протоколу, на сервері не зберігається інформація про клієнта, все необхідне для автентифікації та авторизації міститься в самому HTTP-запиті.

Великою перевагою односторінкових застосунків є незалежність серверної та клієнтської частин, вони можуть розроблятись окремо. Клієнт за необхідності отримує від серверу дані, які динамічно завантажуються на сторінку без її перезавантаження. Переходи між сторінками застосунку здійснюється за рахунок клієнтських скриптів, що випливає у позитивний користувацький досвід через відсутність перезавантажень.

Загалом, підхід SPA з кожним роком стає все більш поширеним, з’являються нові інструменти його реалізації. Найбільш поширеними є наступні фреймворки: React, Angular, Vue.js.

Серед проблем, що можуть виникнути при даному підході виділяють складність SEO-оптимізації через те, що пошукові роботи не прилаштовані до індексації динамічного контенту. Також проблемою є те, що зі збільшенням складності клієнтської логіки зростає необхідність в ресурсах, а отже, падає швидкість. Проте постійна оптимізація фреймворків поступово зводить проблеми нанівець.

2 Архітектура клієнтської частини веб-застосунку на Vue.js

2.1 Історія створення Vue.js

Ідейним натхненником фреймворку є працівник компанії Google Еван Ю. Коли у нього виникла необхідність створити прототип складного інтерфейсу без повторюваного HTML коду, тоді зародилася ідея Vue.js. За словами засновника, Vue.js взяв найкраще, що було в AngularJS, та запакував це в легковагу оболонку.

Перший реліз відбувся в лютому дві тисячі чотирнадцятого року. З тих пір фреймворк значно еволюціонував та наразі надає функціонал достатній для реалізації складних веб-застосунків.

Поточна версія фреймворку – «2.6 Macross». Vue.js підтримується засновником та інтернаціональною командою розробників.

2.2 Система компонентів Vue.js

Кожен клієнтський застосунок, написаний на Vue.js містить в собі екземпляр Vue, який є коренем. Він створюється за допомогою виклику функції new Vue({//options}) з певними параметрами або без них.

Окрім кореня, програма зазвичай складається і з інших екземплярів Vue, які називаються компонентами. Кожен компонент має унікальне ім’я та може бути використаний багато разів [2].

Компонент складається з наступних секцій:

- Шаблон. Може бути звичайна HTML розмітка з використанням певних директив, специфічних для Vue (v-if, v-for, v-on…). Загорнуті всередині тегу < template >.
- Скрипти. В цій секції описані ряд функцій та полів, що визначають логіку роботи компоненту. Наприклад, в полі

components зберігається інформація про компоненти, що являються дочірніми компонентами даного. В полі props визначені властивості, що мають бути передані компоненту на етапі ініціалізації. Функція data() містить в собі поля, що зав'язують певні значення, на ключі що містяться в шаблоні за допомогою директиви v-model. Поле computed містить в собі обчислювальні властивості, значення яких постійно оновлюється «на льоту». Загорнуті в тег < script >.

- Стилі. Набір певних CSS стилів, що використовуються в шаблоні. Загорнуті всередині тегу < style >.

Таким чином, Vue застосунок представляє собою дерево вкладених компонентів. Логічно побудована ієрархія компонентів – запорука швидкої та легко-масштабованої програми.

2.3 Життєвий цикл екземпляру Vue

В процесі роботи екземпляр Vue проходить через ряд етапів життєвого циклу. Схематично життєвий цикл виглядає так [3]:

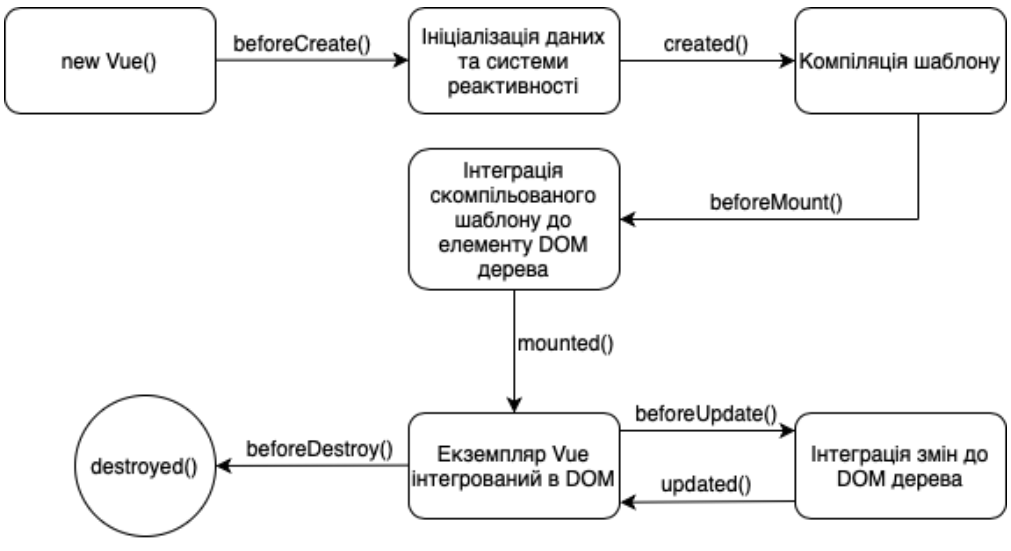

Рисунок 2.3.1 – Життєвий цикл екземпляру Vue

Як позначено на схемі, Vue екземпляр проходить велику кількість етапів ініціалізації: створення даних, компіляція шаблону, інтеграція шаблону до DOM дерева. Між кожним з цих етапів викликаються певні методи – так звані хуки життєвого циклу.

Слід зазначити, що розробник має можливість додати власну логіку, використовуючи будь-який хук життєвого циклу Vue екземпляру. Ця можливість робить систему гнучкою й адаптивною до змін та найвибагливіших функціональних вимог.

2.4 Реактивна парадигма Vue.js

Реактивне програмування – парадигма, орієнтована на потоки даних та розповсюдження змін. Тобто існує модель взаємодії компонентів системи, при якій компоненти нижнього рівня автоматично розповсюджують зміни до компонентів верхнього рівня.

Простий приклад:

В імперативній парадигмі присвоєння $x = y + z$ буде означати, що змінній x буде присвоєно значення результату виконання виразу $y + z$. В майбутньому змінні y та z можуть набувати інших значень без будь-якого впливу на значення змінної x.

Натомість при реактивній парадигмі значення змінної x буде автоматично перераховані, зважаючи на нові значення змінних y та z.

Саме за рахунок реактивності модель побудови веб-застосунків, використовуючи підхід SPA, стала можливою. Фреймворк Vue.js інкапсулює методи реалізації реактивної парадигми всередині [4]:

Коли JavaScript об’єкт передається в екземпляр Vue в якості опції data, Vue перетворює всі його поля в пари getter/setter, використовуючи JavaScript функцію Object. defineProperty. Ці пари не видні користувачу,

проте саме їх наявність є внутрішнім механізмом відслідковування змін даних.

До кожного екземпляру компонента поставлений у відповідність пов’язаний із ним екземпляр спостерігача. Спостерігач відмічає всі поля, що використовуються при генеруванні компонента. Коли поле компонента зазнає змін, викликається setter, який повідомляє спостерігачу про зміну. Спостерігач ініціює повторну генерацію компонента.

Зважаючи на дану особливість реалізації реактивної парадигми, цілком очевидним є висновок, що задля швидшої роботи системи необхідно диференціювати великі компоненти на малі незалежні компоненти, які відповідають принципу єдиної відповідальності.

2.6. Vuex – шаблон керування станом

Компоненти залежать від стану. Стан – це певні дані, що використовуються в застосунку. Коли застосунок невеликий за кількістю компонентів, то не виникає проблем із передачею певного стану від одного компоненту до іншого. Якщо два або більше компоненти залежать від однієї змінної, то розробнику складніше контролювати коректне відображення цього стану в усіх областях його видимості. Звісно, Vue.js надає можливості передачі зміни стану від дочірнього компоненту батьківському за допомогою оператору $emit. І це не єдиний варіант зберегти функціональність програми. Проте є спеціальна технологія, призначена саме для керування станом застосунку з одного місця.

Vuex є централізованим сховищем даних для всіх компонентів застосунку. Vuex керує станом і гарантує, що стан може бути змінений тільки із середини Vuex. Кожен компонент за необхідності робитиме запит до сховища даних та передаватиме зміни даних назад у сховище [5].

Архітектурний підхід Flux – ідейний натхненник Vuex. Основною характеристикою цього підходу є чітке виділення концепції керування станом, як ізольованою системою, в якій прописані механізми підтримки незалежності між представленням (відображення стану) та станом задля кращою структурованості коду. Але на відміну від інших реалізацій цього підходу (наприклад Redux), Vuex створений у вигляді бібліотеки, спеціально налаштований для Vue.js з урахуванням його системи реактивності.

Основні поняття Vuex

- Поняття стану. Vuex зберігає єдине дерево стану – один об’єкт слугує єдиним джерелом стану для всіх компонентів програми. Тобто певна змінна, що зберігає своє значення, може бути викликана та застосована в будь-якому компоненті. Викликати певний стан можна за допомогою функції this.$store.state.{name}. Якщо ж компонент звертається до великої кількості полів, що зберігаються в сховищі, то можна скоротити код та викликати функцію mapState({//options}) в секції скриптів відповідного компоненту.
- Поняття геттеру. Коли виникає необхідність провести певні обчислення над об’єктом, що зберігається в сховищі, у кількох компонентах, то задля зменшення дублювання коду існує механізм геттерів. Вони також знаходяться всередині сховища, але відрізняються від стану тим, що їх значення обчислюються кожен раз при зміні значення вхідного параметру. Викликати геттер всередині компоненту можна за допомогою функції this.$store.getters.{name}. Якщо ж компонент звертається до великої кількості геттерів, то можна скоротити код та викликати функцію mapGetters({//options}) в секції скриптів відповідного компоненту.

- Поняття мутації. Мутація – єдиний спосіб змінити стан сховища. Кожна мутація має стрічковий тип та функцію-обробника. В обробнику відбувається зміна стану, що був переданий всередину функції першим аргументом. Щоб викликати мутацію, необхідно викликати функцію store.commit('{mutationType}'). Більше того, в мутацію можна передати другий параметр – навантаження, який часто іменується як payload. Ще одна особливість мутацій – вони синхронні. Їх можна викликати всередині компонентів за допомогою функції this.$store.commit('{mutationType}'). Якщо ж компонент звертається до великої кількості мутацій, то можна скоротити код та викликати функцію mapMutations({//options}) в секції скриптів відповідного компоненту. Проте найчастіше мутації викликаються з дій.
- Поняття дії. Дії подібні до мутацій, проте мають відмінності. Вони зазвичай використовують асинхронні операції, а також не змінюють стан напряму, а викликають мутації для виконання цієї операції. Дії використовують для отримання чи зміни даних на сервері. Дії запускаються за допомогою методу this.$store.dispatch('{actionType}'). Проте також можна заявити про декілька дій всередині компоненту, використовуючи функцію mapActions({//options}) в секції скриптів відповідного компоненту.

Отже, Vuex надає інтуїтивно-зрозумілий та ефективний механізм керування станом застосунку, що надає змогу скомпонувати важливі елементи системи в одному сховищі. Інкапсульованість стану в цьому сховищі та неможливість змінити його поза межами сховища робить цю

систему простою та водночас безпечною для розробки складної логіки веб-застосунку.

Принцип роботи Vuex схематично виглядає наступним чином:

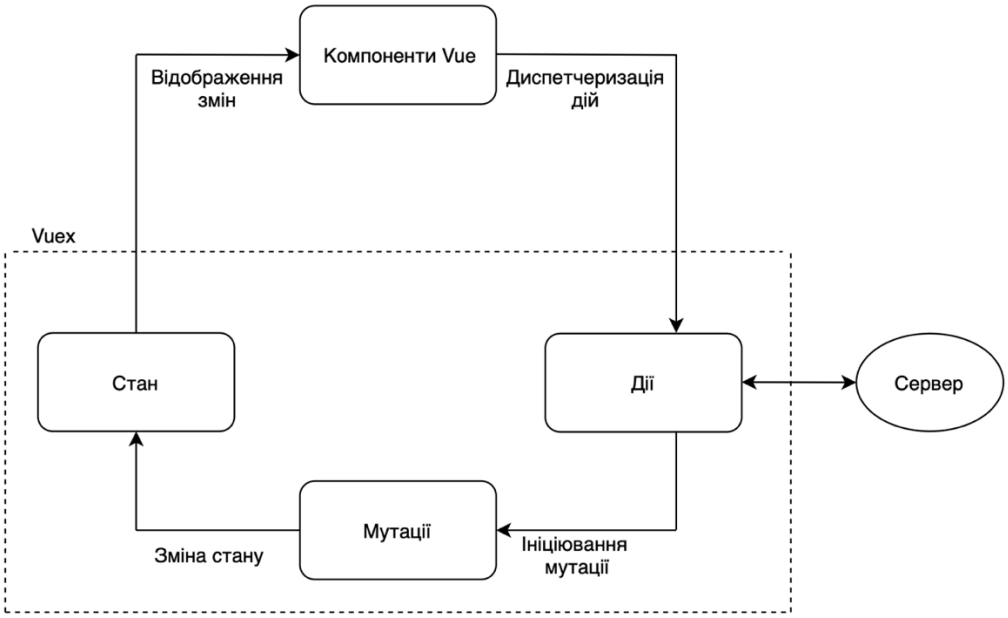

Рисунок 2.6.1 – Принцип роботи Vuex

2.7 Vue Router – механізм маршрутизації

Vue.js, як широко-функціональний фреймворк для клієнтської частини веб-застосунку мав би надавати можливість браузерної маршрутизації між сторінками застосунку. Не зважаючи на те, що по факту ми маємо справу з односторінковою програмою, фреймворк не обділяє користувача можливістю здійснювати навігацію в межах адресної стрічки браузера.

Як відомо, програма на Vue.js – набір вкладених компонентів. Додаючи Vue Router, ми співставляємо компоненти з маршрутами та вказуємо Vue Router, де відображати той чи інший компонент [6].

Найпростіший приклад використання Vue Router виглядає так:

```
<template>
…
<router-link to="/somepage"> Перейти на somepage </router-link>
//тут відобразиться компонент після кліку на відповідний роут
<router-view></router-view>
…
</template>
<script>
…
const routes = [{path: ="/somepage", component: Somepage}]
const router = new VueRouter({ routes })
…
</script>
```

В ході інтеграції маршрутизатора в програму часто з’являється потреба звертатись до нього в коді скрипту. Наприклад, для фізичного переходу на іншу сторінку (насправді, для зміни відображеного компоненту). Звісно, Vue.js надає простий і надійний інструментарій для цього. Щоб перейти на головну сторінку застосунку достатньо всередині тегу < script > звернутись до роутера та викликати метод push з відповідним параметром: $this.router.push('/').

Наявність зручного механізму маршрутизації значно полегшує роботу розробнику для побудови односторінкового застосунку, що надає коректний та приємний досвід користувачу системи.

3 Огляд додаткових можливостей на базі фреймворку Vue.js

3.1 Vue Resource

Окрім того стандартного функціоналу, що надає Vue.js з коробки, існує низка систем, які можуть бути успішно інтегровані в застосунок, побудований з використанням цього фреймворку.

Наприклад, кожен веб-застосунок має серверну частину. І клієнтська частина тим чи іншим чином бере дані з серверної частини, використовуючи HTTP запити. Часто застосунки на Vue.js побудовані з використанням RESTful архітектури – клієнт та сервер фактично незалежні, з кожним запитом до серверу передається вся необхідна інформація про клієнта. Великою перевагою даного архітектурного рішення є можливість виконувати запити асинхронно, без фактичного переривання роботи клієнтської частини.

Vue Resource – відкрито-сирцевий плагін, написаний групою ентузіастів спеціально для фреймворку Vue.js. Даний плагін надає сервіси для виконання мережевих запитів та обробки відповідей з використанням XMLHttpRequest або JSONP [7].

Vue Resource має низку переваг, що роблять його дуже зручним у використання та водночас функціональним. По-перше, Vue Resource дуже легкий – всього 14 KB. По-друге, він підтримує інтерцептори для запитів та відповідей, що розширює поле дій для розробника. По-третє, даний плагін підтримує обидві версії фреймворку Vue та усі сучасні браузери. По-четверте, завдячуючи підтримці Promise Api, асинхронні запити до серверу виконуються в максимально зрозумілій для розробника формі.

Ініціалізацію ресурсу виглядає наступним чином:

```
const resources = Vue.resource('resource{/id}')
```

Змінна id – опціональна, її можна вставити при виконанні одного з наступних запитів:

- resources.save({}, resource) – відповідає за надсилання HTTP Post запиту. Приймає об’єкт в якості параметру.
- resources.update({id}, expense) – метод, що відповідає за надсилання HTTP Put запиту. Приймає об’єкт та ідентифікатор об’єкту в якості параметрів, згідно до RESTful архітектури.
- Resources.remove({id}) – метож, що відповідає за надсилання HTTP Delete запиту. Приймає в якості параметру ідентифікатор об’єкту.

Звісно, це не повний перелік методів бібліотеки Vue Resource. Навіть більше, вищенаведені методи теж можна змінити в плані кількості та наявності параметрів.

Для надсилання асинхронних запитів на сервер існує не тільки інструмент Vue Resource. Більш поширена є бібліотека Axios. Наявні й інші бібліотеки. Ця гнучкість вибору надає приємний головний біль розробникам.

3.2 Vuetify

Якщо мова йде за інтерфейс програми, то Vuetify – бібліотека номер один для клієнтського застосунку, написаному на Vue.js [8]. Вона надає можливість розробнику створювати приємний та функціональний користувацький інтерфейс, згідно з принципами Google Material Design.

Vuetify надає сотні готових стилізованих компонентів: форми, кнопки, банери, текстові блоки. Більше того, ці компоненти можна

підлаштовувати під поточні потреби за допомогою широкої палітри атрибутів, які можна змінювати.

Опанувати дану бібліотеку можна за найкоротші терміни завдяки якісно оформленій документації з прикладами та «пісочницями» для кожного компоненту.

Наявність такої бібліотеки значно розв’язує руки розробникам, що прагнуть зробити сучасний мінімалістичний дизайн свого застосунку в найкоротші терміни. Vuetify позбавляє необхідності користуватися оголеним HTML, писати власні класи стилів чи довжелезну розмітку.

Ще однією беззаперечною перевагою цієї бібліотеки є адаптивність із коробки. Тобто будь-який компонент Vuetify відображається коректно як на мобільних пристроях, так і на екранах комп’ютерів.

Підсумовуючи, Vue.js настільки стрімко увірвався на арену сучасних JavaScript фреймворків, що стало справжньою несподіванкою для всіх великих гравців на ринку. Незважаючи на те, що конкуренти в обличчі React та Angular підтримуються такими гігантами як Facebook та Google відповідно, Vue.js з кожним роком стає все більш популярним через свою простоту та велику кількість відкрито-сирцевих плагінів та бібліотек, які написані не під вимоги тих чи інших великих корпорацій, а для якомога більшої зручності у використанні для розробників. Така широка функціональна палітра робить Vue.js придатним до найвимогливіших та різноманітніших веб-систем.

Незалежне співтовариство, зацікавлене у розвитку Vue.js, робить вагомі внески в розбудову фреймворку. Vue Resource та Vuetify – лише два з багатьох прикладів політики утримувачів фреймворку щодо збереження простоти для користувачів-розробників та відкритості до нових ідей та розширення можливостей технології.

4 Реалізація практичної частини роботи

4.1 Опис предметної області

Контроль власних фінансів – необхідна річ в сучасному світі. Наразі існує занадто багато зовнішніх чинників, що змушують робити витрати свідомо чи підсвідомо. Тим легше стає потрапити в пастку щурячих перегонів. Щоб цього уникнути, необхідно обрати спосіб контролю своїх доходів та витрат.

Деякі люди все записують у фізичні носії: блокноти, щоденники, зошити. Проте вони не завжди є під рукою, тому для більш ефективного результату було б раціонально мати записник в цифровому форматі, щоб мати доступ з комп’ютеру чи смартфону.

Наразі існує багато програмних рішень цієї проблеми, проте всі вони чимось відрізняються та водночас кожна має свого користувача.

Проте мало хто з розробників цих рішень керується думкою, що успіх полягає в максимальній простоті при максимальному функціоналі. Зазвичай програми перенавантаженні зайвими деталями, через які люди просто заплутуються, що та як їм треба робити.

Фреймворк Vue.js та бібліотека стилів Vuetify ідеально підходять для розробки застосунків такого типу. При лаконічному та зрозумілому дизайні від Vuetify можна використати весь функціонал та переваги реактивного односторінкового застосунку Vue.js.

Враховуючи той факт, що Vuetify надає властивість адаптивності із коробки, фінальний програмний продукт можна з успіхом використовувати, як з комп’ютеру, так і з мобільного пристрою будь-якого розміру без втрати гармонійного відображення даних та функціональності.

4.2 Архітектура застосунку

Financy – програмне уособлення найкращих ідей та можливостей, наданих фреймворком Vue.js. Це односторінковий застосунок, побудований згідно до архітектури REST. Тобто клієнтська частина незалежна від серверної, все необхідна інформація для виконання запиту міститься в самому запиті.

Процес автентифікації відбувається за допомогою механізму Google Sign-in – усім відома автентифікація через дані електронної пошти компанії Google. На даний момент часу в цій системі зареєстровано щонайменше півтора мільярда користувачів. Тому можна припустити, що абсолютна більшість бажаючих зможе скористуватись застосунком Financy.

Google Sign-in працює за відкритим протоколом авторизації OAuth2. Тобто програма Financy не отримує пароль користувача Google, а при процесі автентифікації користувача, увівши свої дані, він надає серверу Financy так званий authorization grant. Сервер Financy в свою чергу звертається до авторизаційного серверу Google з цим authorization grant та надає певну інформацію про себе. Сервер Google надає токен доступу до ресурсу (даних користувача), який використовується сервером Financy для внутрішнього процесу автентифікації користувача.

Перевагою такого підходу є те, що застосунок не несе відповідальність за збереження чуттєвих даних користувача, а лише користується готовим і прозорим механізмом компанії Google без втрати функціоналу.

Серверна частина реалізована на мові програмування Kotlin із використанням фреймворку для побудови веб-застосунків Spring Boot. Дані зберігаються в системі керування базами даних PostgreSQL.

4.3 Реалізація серверної частини

Для реалізації серверної частини застосунку Financy була обрана мова програмування Kotlin у зв’язці із фреймворком Spring Boot. Саме такий вибір технологій зумовлений тим, що Kotlin є ідейним послідовником однієї з найбільш поширених об’єктно-орієнтованих мов програмування Java, проте має свої переваги і загалом є більш сучасним. Spring Boot – фреймворк для швидкої й інтуїтивно-зрозумілої конфігурації серверу.

Перевагою такої звя’язки є, перш за все, лаконічність коду. Можливості Kotlin та конфігурації Spring Boot, якій здійснюються за допомогою анотацій, роблять код коротким та зрозумілим. При цьому зберігаються всі переваги об’єктно-орієнтованого підходу як: чітке виділення об’єктів реального світу в програмному коді, використання принципів абстракції, інкапсуляції, поліморфізму та наслідування в повному обсязі.

Наприклад, щоб отримати об’єкт витрати за ідентифікатором достатньо прописати два рядки коду в класі контролеру.

```
@GetMapping("{id}")
fun getExpenseById(@PathVariable("id") expense: Expense) = expense
```

На сервері виділені три основні сутності даних: Expense (витрата), Income (дохід), User (користувач). Дані зберігаються базі даних PostgreSQL. Ця система керування базами даних була обрана через низку її переваг: широкий набір стандартних типів, високу продуктивність, наявний механізм підтримки цілісності посилань, необмежений розмір пам’яті.

Загалом, обрані технології цілком задовольнили потреби серверної частини.

4.4 Функціонал застосунку

Метою розробки сервісу для контролю власних фінансів Financy було надати користувачу максимально дружній інтерфейс із інтуїтивно-зрозумілими функціями.

Реалізовані функціонал полягає в наступному:

- Переглядати доходи та витрати поточного місяця в зручній формі та з можливістю розгорнути для отримання деталізації кожного запису.
- Відображення загального балансу та його реактивний перерахунок при зміні даних.
- Відображення балансу за кожною категорією доходу чи витрати та реактивний перерахунок при зміні даних.
- Створити, оновити, видалити витрату або дохід.
- Можливість здійснювати переключення між місяцями з відповідним оновленням відображених даних.
- Неможливість переключитись на місяць, який не має записів, за умови що він знаходиться поза межами часового інтервалу між самим раннім та найпізнішим записами.
- Валідація на коректність суми доходу або витрати та на необхідність обрати категорію при створенні чи оновленні записів.
- Вбудоване сортування записів: пізніша дата згори (в межах групи).
- Маршрутизація між сторінкою графіків та представленням даних у вигляді списку.
- Відображення записів у вигляді графіку: відсотковий показник витрат/доходів кожної категорії, відсоток та сума.
- Механізм автентифікації з використанням Google Sign-in.

4.5 Реалізація клієнтської частини

При реалізації клієнтської частини застосунку Financy були використані всі переваги фреймворку Vue.js та побудованої навколо нього екосистеми.

Програма складається восьми компонентів, які в залежності від дій користувача реактивно змінюються. Незмінним лишається кореневий компонент App. Структура компонентів наведена нижче:

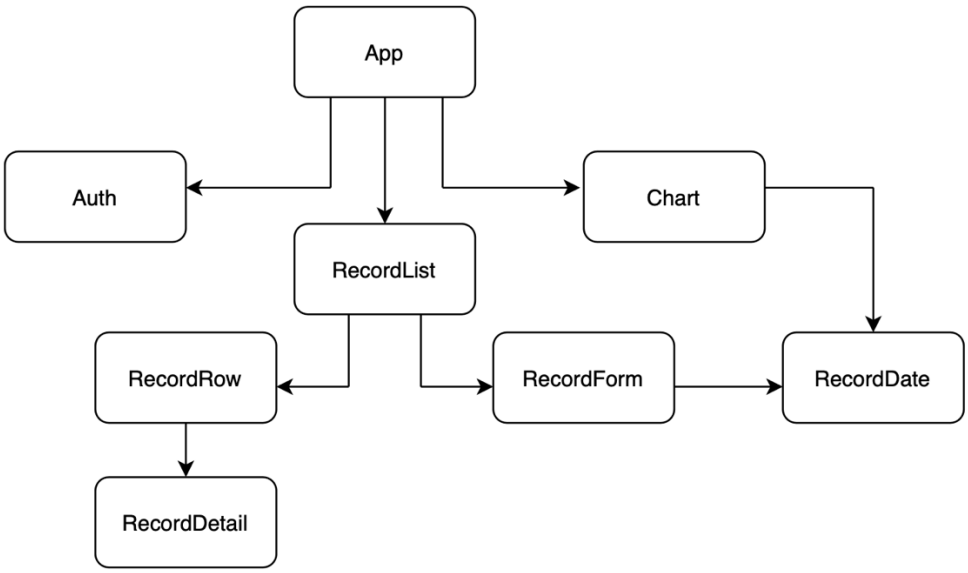

Рисунок 4.5.1 – структура компонентів Financy

Використання компоненту RecordDate в двох інших компонентах наглядно ілюструє механізм повторного використання коду задля уникнення дублювання – те, заради чого фреймворк Vue.js був створений.

Задля досягнення кінцевого результату були використані описані в попередніх пунктах інструменти як: Vuetify, Vuex, Vue Router, Vue Resource. Крім цього, механізм валідації форм реалізований з бібліотекою Vuelidate, графіки зроблені з використанням пакету VueGoogleCharts.

4.6 Інтерфейс користувача

При вході на головну сторінку програми користувачу одразу пропонується пройти процес Google автентифікації.

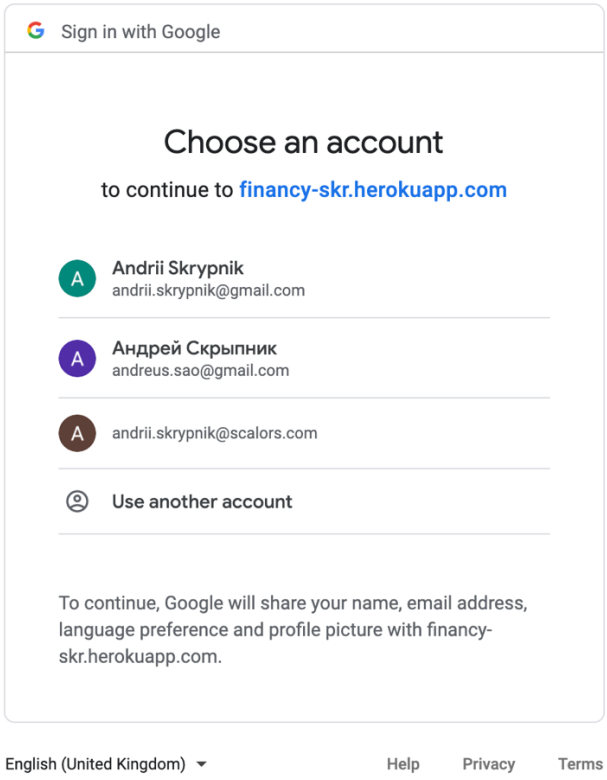

Рисунок 4.6.1 – процес автентифікації

Після успішної автентифікації користувач перенаправляється на головну сторінку застосунку – відображення доходів та витрат у вигляді списку. Якщо користувач щойно перший раз здійснив вхід до застосунку, очевидно, він не матиме ніяких записів. Проте якщо користувач вже активно користується застосунком, то його головна сторінка виглядатиме наступним чином.

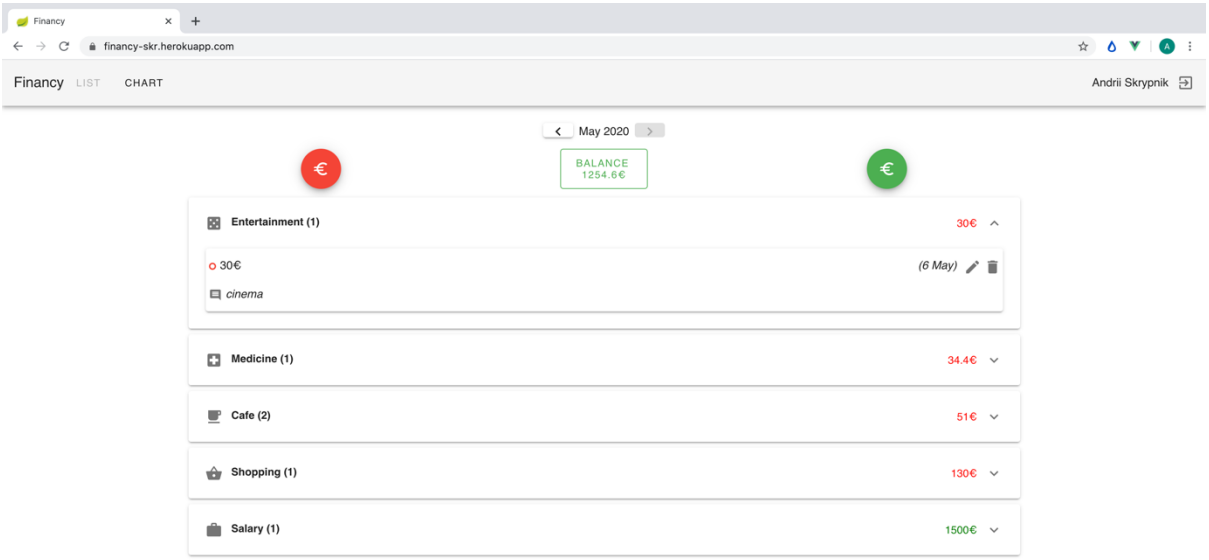

Рисунок 4.6.2 – головна сторінка

В хедері сторінки міститься назва програми, кнопки переходу між представленням даних у вигляді списку та у вигляду графіку, а також ім’я користувача, отримане від серверу Google. Відповідною іконкою позначена кнопка «вийти» з застосунку.

Користувач має змогу побачити згруповані за категоріями відомості про свої доходи та витрати. Якщо це витрата, то число виділятиметься червоним кольором, якщо дохід – то зеленим. Це зроблено задля інтуїтивної зрозумілості наявних чисел. Кожну категорію можна розгорнути, щоб побачити деталізацію усіх записів цієї категорії за поточний місяць. Видно суму, дату, коментар кожного запису. Навпроти

кожного запису наявні кнопки «редагувати» ти «видалити», позначені відповідними іконками.

Також користувачу надається можливість створити нові записи, натиснувши відповідні кнопки справа та зліва від відображеного загального балансу. Якщо баланс позитивний, він відображений зеленим кольором, якщо негативний – червоний, якщо нуль – сірим. Крім цього, є кнопки переходу на попередній та наступний місяці із відповідною реактивною зміною відображених даних.

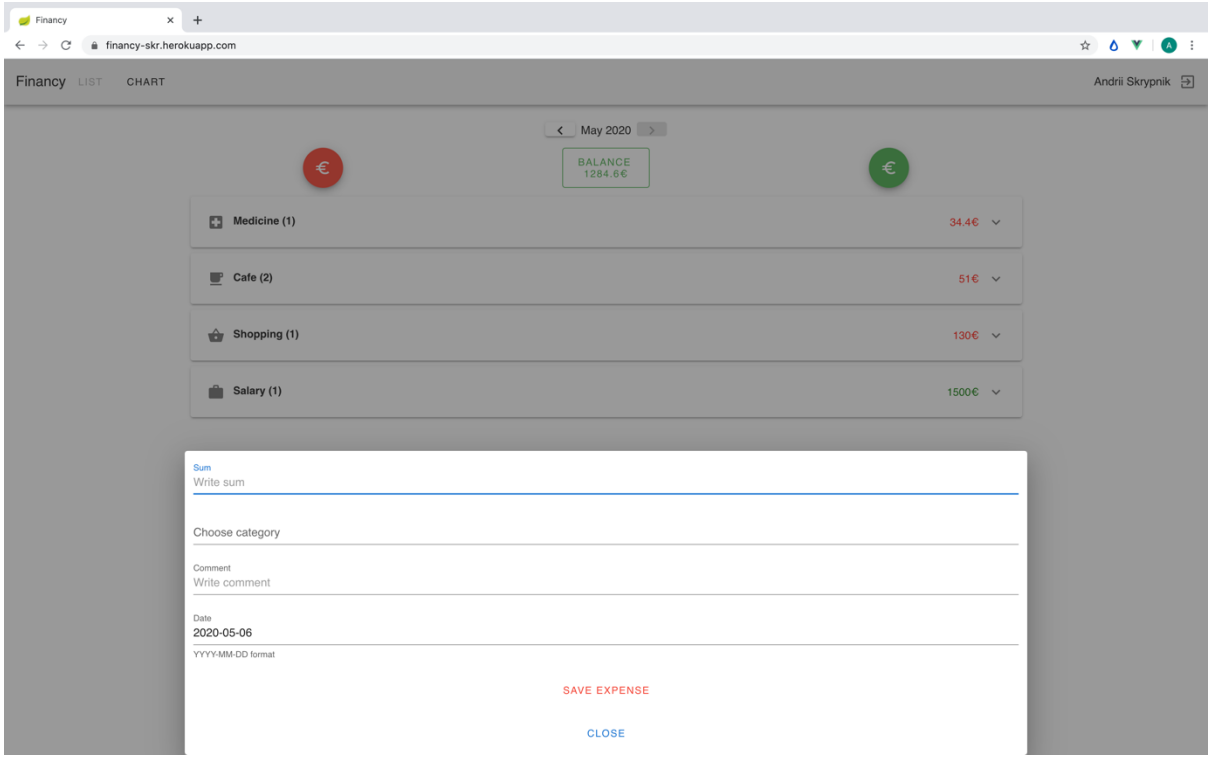

Рисунок 4.6.3 – створення витрати

При створенні витрати чи доходу користувачу пропонується ввести суму, обрати категорію, написати коментар (за бажанням) та обрати дату. За замовченням, буде обрана поточна дата, проте користувач не обмежений у виборі: можна обрати дату в минулому чи в майбутньому. Якщо він натисне на дату в певному місяці, в якому ще не було жодного запису, застосунок реактивно перебудовується та буде відображати

відповідний місяць, згідно до дати останнього запису. Наявне сортування записів за спаданням, починаючи від найпізнішого запису місяця.

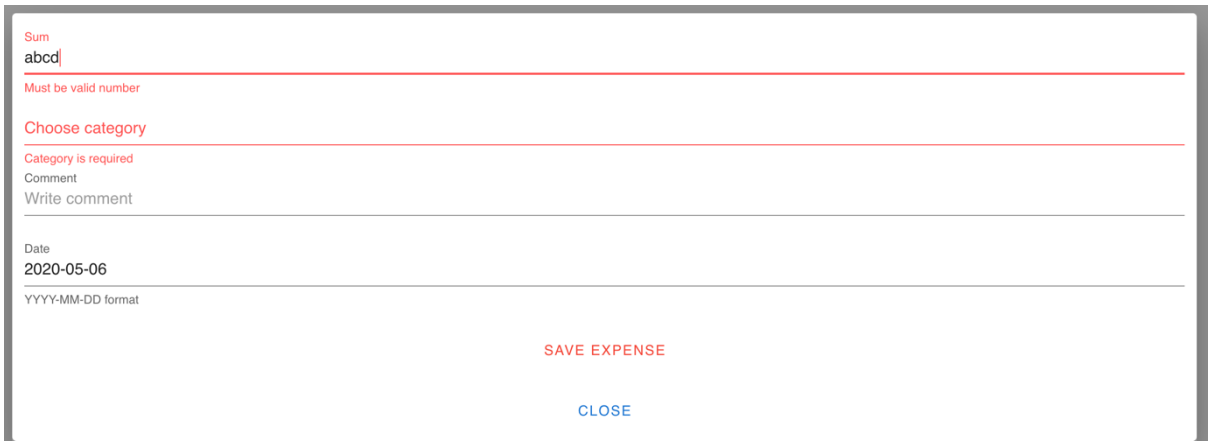

Рисунок 4.6.4 – валідація форми

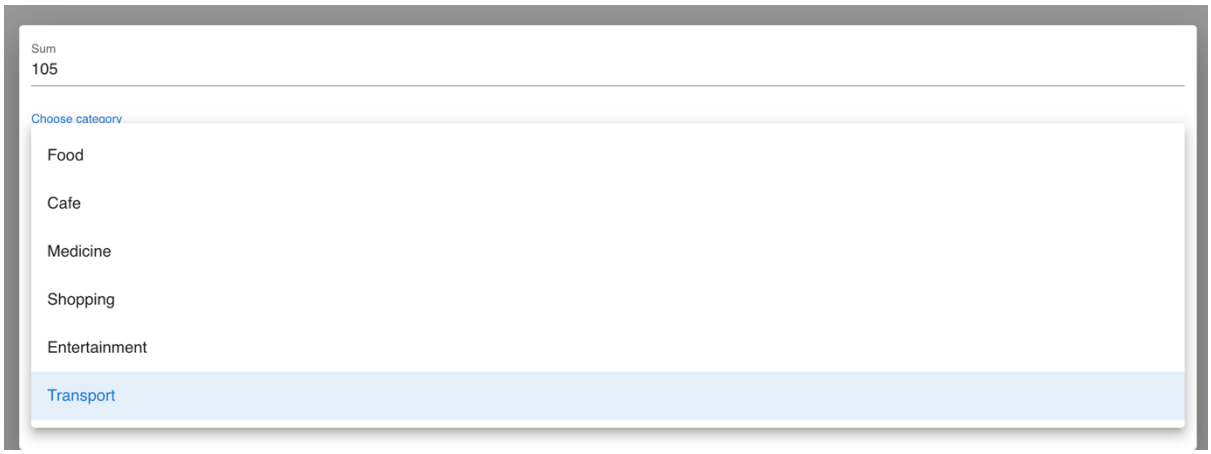

Рисунок 4.6.5 – вибір категорії витрат

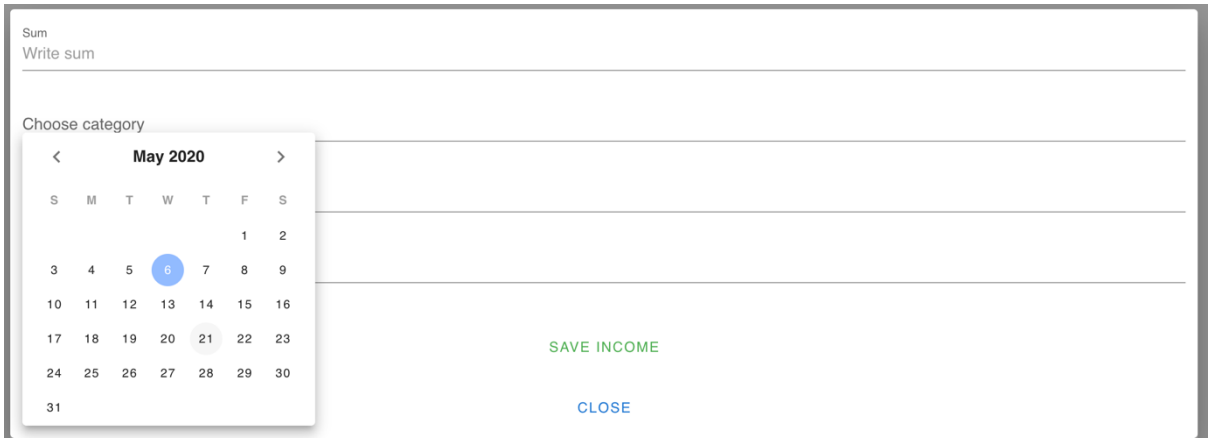

Рисунок 4.6.6 – вибір дати запису

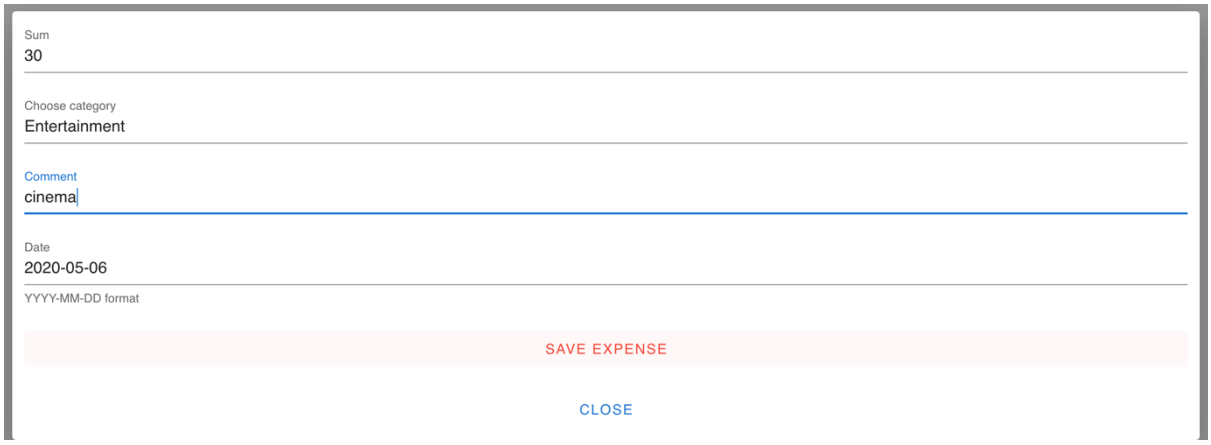

Рисунок 4.6.7 – введені дані форми створення витрати

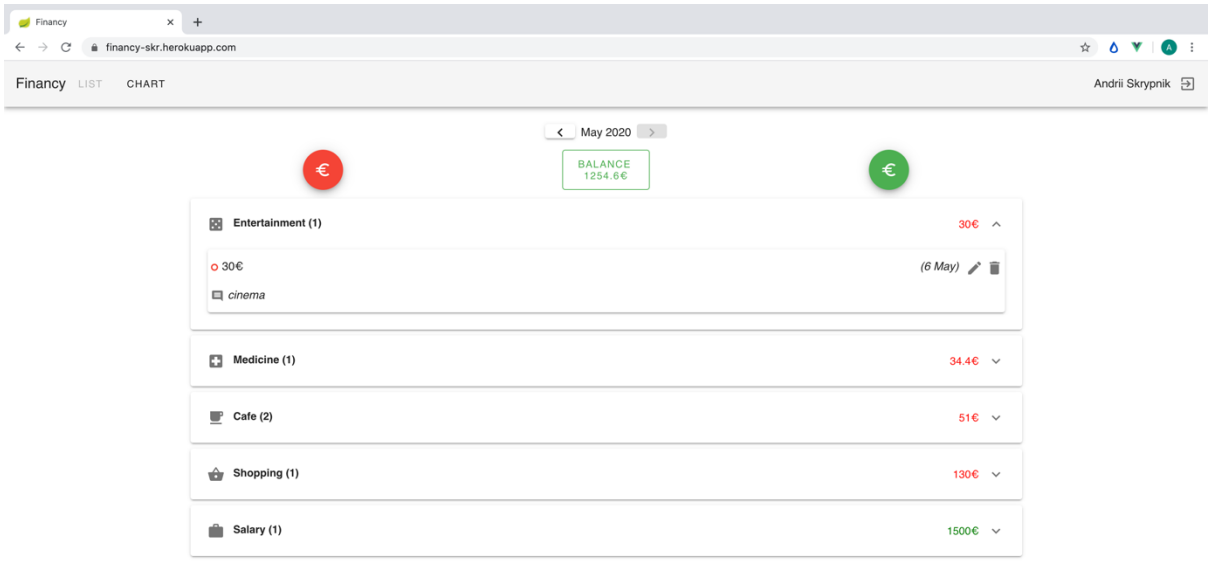

Рисунок 4.6.8 – результат збереження витрати

Дуже корисною є функція графічного відображення витрат і доходів по місяцях. Користувач має змогу оцінити свої фінансові показники у вигляді кругової діаграми. Для цього достатньо натиснути на кнопку «Chart» в хедері застосунку. При наведенні на певний сектор діаграми, буде відображена сума в абсолютній величині витрат або доходів за відповідною категорією у відповідний місяць та відсоток цієї суми від загальної суми витрат або доходів. При графічному відображенні даних залишається можливість переключатись між місяцями, щоб бачити показники по кожному відповідному місяцю.

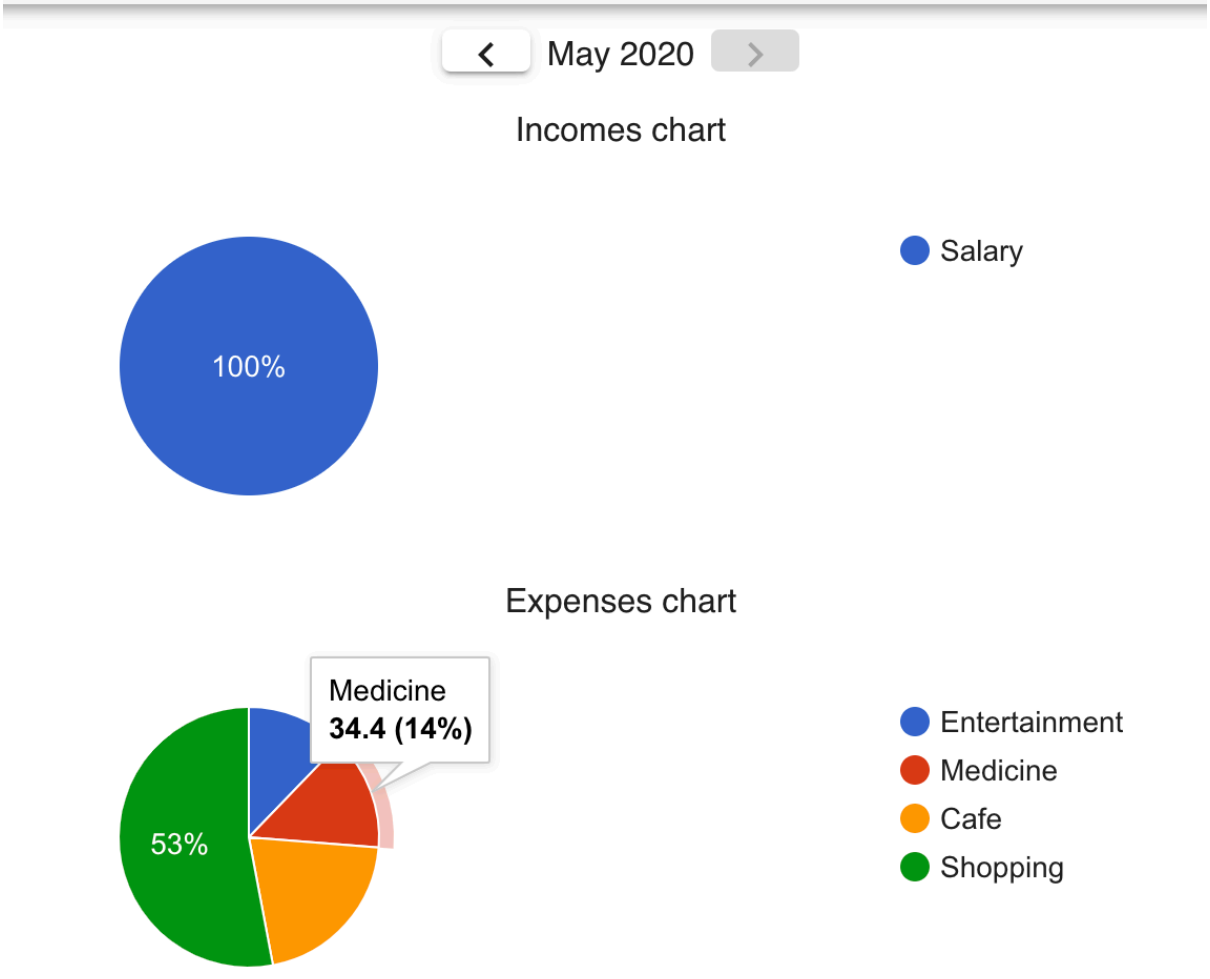

Рисунок 4.6.9 – графічне відображення даних

Як було згадано раніше, фреймворк Vue.js у зв’язці із бібліотекою Vuetify надає адаптивний інтерфейс із коробки. Тобто всі дані, кнопки, форми, графіки відображаються на мобільних пристроях в дружньому, адаптивному вигляді.

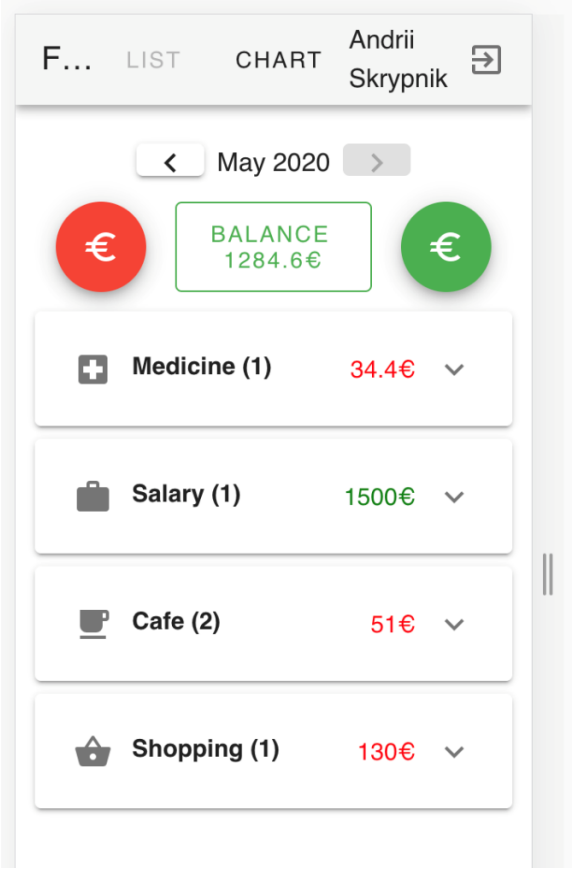

Рисунок 4.6.10 – інтерфейс мобільного пристрою

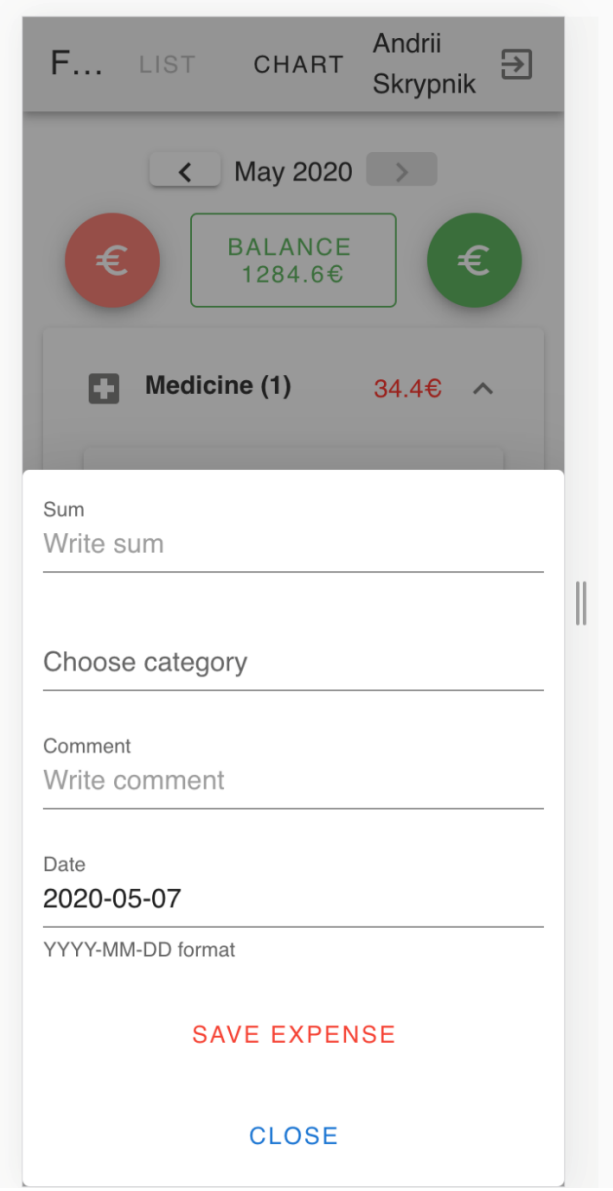

Рисунок 4.6.11 – форма збереження витрати на мобільному пристрої

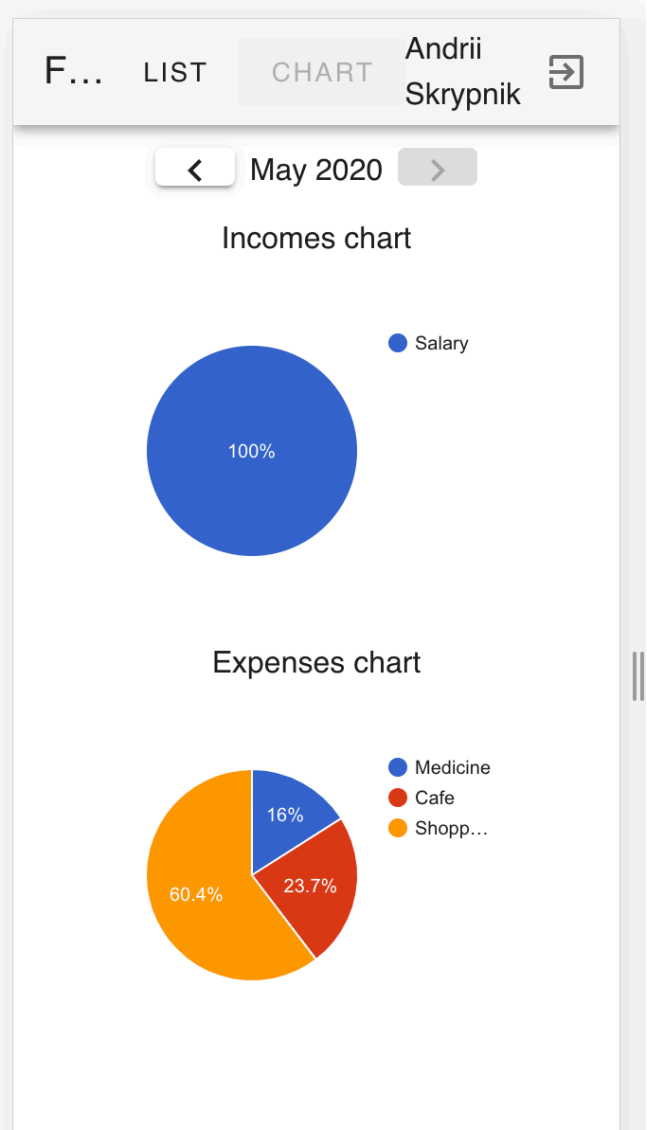

Рисунок 4.6.11 – відображення графіків на мобільному пристрої

Висновки

В ході виконання курсової роботи були досліджені основні аспекти розробки клієнтського застосунку з використанням фреймворку Vue.js. Були проаналізовані переваги програмної архітектури, запропонованої фреймворком, а також екосистемою, сформованою навколо нього.

Отже, Vue.js був створений групою ентузіастів як зручний та легковагий інструмент побудови односторінкових клієнтських застосунків. Фреймворк не був і не є розробкою певної великої компанії, його підтримує спільнота з усього світу, яка прагне оптимізувати співвідношення якості отриманого результату до кількості написаного коду до якомога більшого значення, що є беззаперечною перевагою Vue.js.

Система компонентів фреймворку та вбудована реалізація реактивної парадигми дозволяють реалізувати в повному обсязі концепцію односторінкового застосунку з прозорою й очевидною логікою роботи, що може бути опанована розробником в найкоротші терміни. Внутрішні інструменти Vuex та Vue Router вирішують проблеми глобального сховища даних та маршрутизації між сторінками, які насправді являються різними реактивно сформованими комбінаціями компонентів.

Більше того, така відкритість фреймворку до спільноти дозволяє вбирати найкращі ідеї у вигляді реалізації додаткових бібліотек з метою розширення функціоналу Vue.js. Наприклад, бібліотека Vuetify надає сотні готових стилізованих компонентів, побудованих згідну принципів Google Material Design, а бібліотека Vue Resource надає можливість виконати асинхронні запити на сервер буквально в два рядки коду.

Описані переваги фреймворку практично підтверджені у вигляді реалізованого програмного застосунку Financy.

Список використаних джерел

1. https://huspi.com/blog-open/definitive-guide-to-spa-why-do-we-need-single-page-applications
2. https://www.weareframework.co.uk/component-communication-vuejs/
3. https://vuejs.org/v2/guide/instance.html
4. https://vuejsdevelopers.com/2017/03/05/vue-js-reactivity/
5. https://vuex.vuejs.org/guide/
6. https://medium.com/better-programming/vuejs-routing-with-vue-router-1548f94c0575
7. https://github.com/pagekit/vue-resource
8. https://vuetifyjs.com/en/