МІНІСТЕРСТВО ОСВІТИ І НАУКИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

ВИПУСКНА РОБОТА

на тему*:*
«Інформаційна система відображення метео-даних»

Завідувач
Випускаючої кафедри **Довбиш А.С.**

Керівник роботи **Ободяк В.К.**

Студента групи ІНз-61с **Литовченко О.Є.**

СУМИ 2020

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

КАФЕДРА КОМП'ЮТЕРНИХ НАУК

Затверджую_____

Зав. кафедрою Довбиш А.С.

“_____” _____2020 г.

ЗАВДАННЯ

до випускної роботи

Студента четвертого курсу, групи ІНз-61с спеціальності “Інформатика” заочної форми навчання Литовченка О.Э.

Тема: “ Інформаційна система відображення метеоданих”

Затверджена наказом по СумДУ

№_____ от _____2013 г.

Зміст пояснювальної записки: 1) Огляд існуючих рішень; 2) постановка завдання й формування завдань дослідження; 3) Вибір середовища розробки; 4) Розробка проекту програми.

Дата видачі завдання “_____”_____2020 г.

Керівник випускної роботи _____Ободяк В.К.

Завдання приняв до виконання _____

РЕФЕРАТ

Записка: 56 стор., 17 рис., 5 табл., 1 додаток, 19 джерел.

Об’єкт дослідження — процесс відображення метеоданих.

Мета роботи — розробити інформаційний веб-сервіс прогнозу погоди для можливості перегляду метео-данних у погодинному та потижневому форматі. Пошук метеозведення по назві міста або населеного пункту.

Методи дослідження — метод емпіричного дослідження.

Результати — розроблений інфомаційний веб-додаток перегляду метеоданих, що включає оперативний пошук метеозведеннь за введеним містом/країною, відстежування інформації у погодинному або потижневому форматі та зручний інтерфейс (UserFriendly-Interface). Обране інтегроване середовище розробки (Visual Studio Code). Обраний метод створення веб-додатку (Vue.js Framework + Node Package Manager). Було поєднано описані вище методи рішення в одну інформаційну систему та розроблене програмне забезпечення для можливості з’ясування погодних умов в конкретному місці. Розроблена інформаційна система була протестована.

МЕТЕОСЕРВІС, ВЕБ ДОДАТОК, ІНТЕГРАЦІЯ ЗОВНІШНІХ REST API, ГРАФІЧНИЙ ІНТЕРФЕЙС, ПОРІВНЯЛЬНИЙ АНАЛІЗ.

ЗМІСТ

ВСТУП

Метеосервіс це інформаційний або науково-дослідний веб додаток, що здійснює вивчення, систематизацію та відображення погодних умов в зручному для користувача інтерфейсі.

У житті кожної людини погода відіграє значну роль – перш ніж вирушити у дорогу, вийти на роботу чи прогулянку, ми дивимося прогноз погоди. Спочатку в якості інформаційної довідки використовувся телевізор, але з часом популярності набули онлайн метеосервіси та так званий «швидкий» пошук інформації в інтернеті. Поступово на ринку почалась конкуренція за лідерство у сфері онлайн – сервісів по погоді. Кожному з нас приємно бачити у своєму смартфоні зручний інтерфейс показу інформації, яка нас цікавить, тому при розробці будь-якого онлайн інтерфейсу важливо продумати зручний дизайн та функціонал сервісу.

Завдяки інтернету ми маємо можливість доступу до великої кількості онлайн-додатків, які збирають та накопичують інформацію з метеостанцій, але не мають чи мають незручний інтерфейс.

Відкритий для кожного користувача комп'ютера метеосервіс, і не один, а десятки і сотні - сьогоднішня реальність.

Сучасні метеосервіси використовують практику співпраці с метеостанціями, які накопичують інформацію про погоду, для створення віртуального середовища, що відбивається на якості сприйняття та засвоєння інформації.

Робота присвячена розробці та створенню інформаційної системи онлайн метеосервісу. Зокрема створення додатку, що відображає на своїх сторінках погодні умови у інтерактивному та детальному вигляді.

1 ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

1.1 Огляд реалізації існуючих метеосервісів.

Метеосервіси присвячені наданню глобального прогнозу погоди та змісту погоди для веб-сайтів, підприємств та туристичної галузі. Про їх створенні мріяли ще на зорі інтернету, але тільки в останні роки завдяки вдосконаленню технологій стали з'являтися красиві і інформативні ресурси.

Незалежно від того, чи шукаєте ви інформацію про найновіші гірськолижні умови, морські умови, звіти про погоду або щоденні прогнози для міст, багато інформаційних сервісів постійно і наполегливо працюють, щоб надати вам точні та сучасні звіти про погоду. Усі прагнуть доставляти точні та сучасні звіти про погоду користувачам у всьому світі, чи вони шукають щоденні оновлення погоди, звіти про серфінг, прогнози лиж або просто зацікавлені в погоді. На багатьох сторінках різних блогів погоди та у соціальних мережах також розміщена інформація про сезонні явища, дискусії про глобальне потепління та статті про інші цікаві елементи погоди та клімату.

Поняття «метеосервіс» містить в собі збірку різного роду Web-сторінок. Вони повинні розташовуватися на одному або декількох Web-серверах. Подібні сторінки містять в собі систематизовану інформацію метео-даних. Такого роду метеосервіс можна зробити в варіанті мобільного додатку.

Зазвичай кожний популярний онлайн метео-сервіс надає послуги доступу до власного API та бази данных звітів погоди та геоположень. Ця послуга буваэ безкоштовна, але чаще всього щоб отримати доступ до API треба оформити платну підписку та отримати ліцензований ключ доступу. Розробник має можливість ввести цей ключ до параметрів зверення запиту та отримати повний список інформації погоди у різних форматах:

- XML;
- CSV;
- JSON.

Метеосервіс (або веб-сайт погоди) - тип веб-сайту, який спеціалізується з метою представлення докладних звіті погоди. Всі матеріали, які можуть бути презентовані в подібному сервісі мають різний характер і вид:

- докладні прогнози рівня по годинах на 7 діб вперед;
- узагальнені прогнози по днях на 14 діб вперед;
- короткі прогнози опадів на 2 години вперед з інтервалами по 15 хвилин;
- звіти погоди за місяць і ін.

Метеосервіси з підтримкою інтернет-технологій можуть вирішувати класичні метеорологічні труднощі - зберігання, безпека, забезпечення широкого, швидкого і легкого доступу інформації.

Відмінність подібних онлайн метеосервісів від офлайн ресурсів(газети, телевізор) в тому, що в здібностях онлайн сервісу існує своєчасне оновлення інформації згідно з передовими метеоцентрами та великі здібності в пошуку будь-яких даних.

З розширенням можливостей інтернет-технологій почали виникати індивідуальні веб-сайти погоди. До таких веб-сайтів належить «Гісметео»[1], відкритий в 2000 році.

Існують наступні функції метеосервісу:

- інфомаційна;
- культурно-просвітницька;
- навчальна;
- довідкова.

Метеосервіс можливо застосовувати з метою онлайн чату для узгодження та прийняття рішення щодо можливості проводення тих чи інших заходів, концертів чи ін. Найістотніша частка мережевих ресурсів - це взаємодія, що важливо в даний час для молодого покоління.

Перше місце метео-роботи в мережевій сфері дістається комунікаційному елементу. Веб-сайт погоди – ще й центр спілкування. Він дає можливість будь-якій людині відшукати шлях до нових територій знання, навичкам і виразам.

Для того щоб створити онлайн метеосервіс, досить щоб в наявності були необхідне технічне обладнання (комп'ютерна техніка, вихід в Інтернет) і доступ до бази данних метеоцентрів. На сьогоднішній день будь-яка просвітницька установа, а крім того різні компанії, в тому числі і туристичні в нашій державі мають подібне технічне обладнання. Те, що відноситься до співробітників, то є спеціалізовані громадські плани, де навчають роботі з такою технікою. Тут працюють кваліфіковані спеціалісти, який зацікавлений в даному процесі. При цьому організація діяльності при наявності сьогодні різних форм роботи в дитячому та молодіжному середовищі, таких збір метеоданих, дозволяє збільшити мотивацію участі молодого покоління до прилучення до проектної та дослідницької діяльності.

У сучасному суспільстві технології метео-центрів домоглися значних результатів та показали всю силу технічних досягнень.

Подібні метеосервіси демонструють себе з найкращого боку, так як дають можливості для формування суспільства.

Одне з рішень створення метеосервісу базується на відображенні погодних звітів у стилі «плитки» чи структуровно-послідовної інформації у зручному інтерфейсі в різних форматах (погодинний, потижневий та ін.).

Також як додаток до існуючуго функціоналу іноді запроваджують можливість реєстрації та комментарів, аби користувачі могли обговорювати теми, які їх цікавлять, приймати рішення щодо проведення чи скасування якихось заходів та іншого. Але іноді підхід з реєстрацією все ж таки краще залишити, бо навантаження на онлайн ресурс позначається на продуктивності та швидкості завантаження контенту на сторінки. Існує багато методів боротьби з покращення продуктивності: один з кращих – обирати бізнес-логіку та інструменти розробки згідно з метою розробки(рис. 1.1) [1].

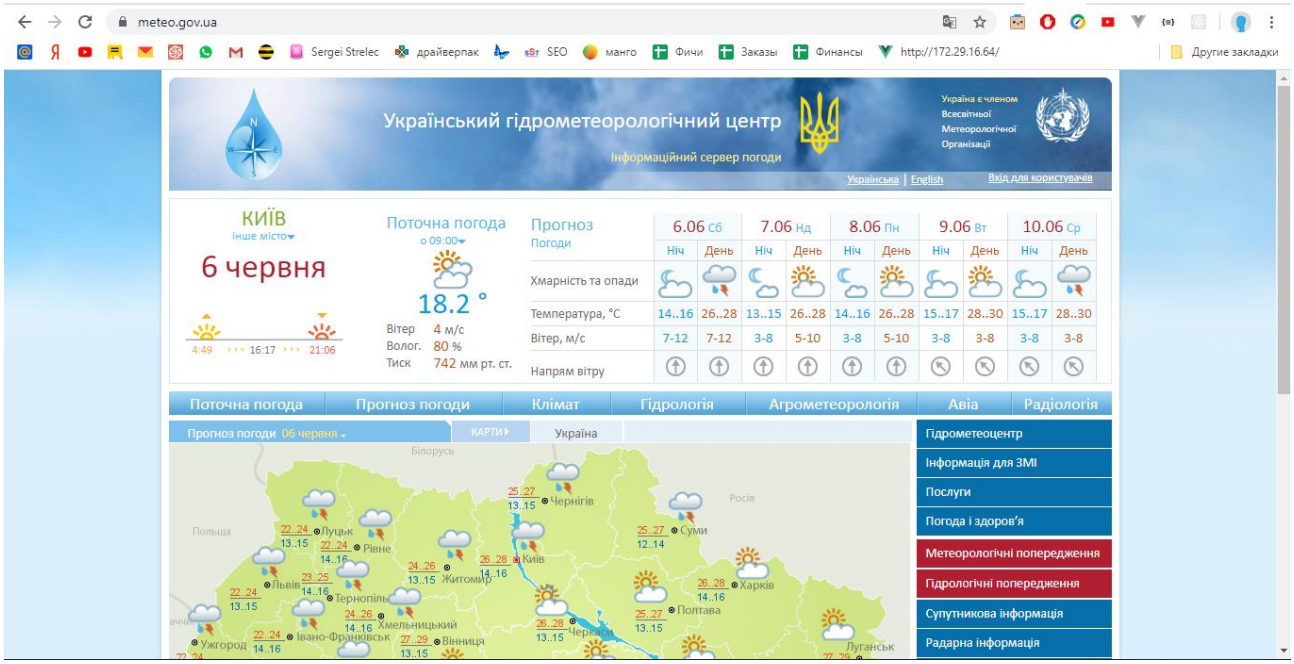

Рисунок 1.1 – Гідрометеорологічний центр

Також деякі метеосервіси демонструють інформацію по погоді в вигляді динамічної карти з різних кутів зору та описом параметрів звіту погоди (рис. 1.2) [1].

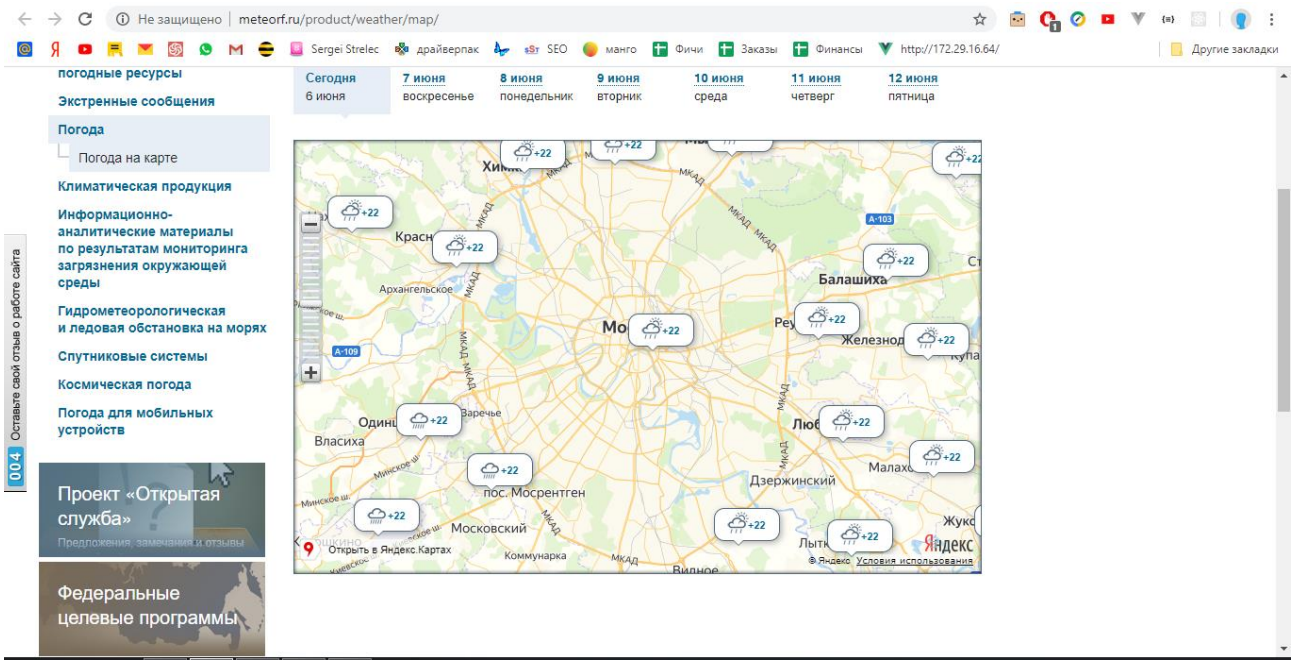

Рисунок 1.2 – Метеосервіс з динамічною картою

Також існує чеська метеорологічна компанія з назвою «The Ventusky»[2]. Чеські розробники створили високоякісний додаток, який чітко відображає

метеорологічні дані з усього світу та дозволяє стежити за розвитком погоди в будь-якому місці Землі. Погода Землі функціонує як взаємозалежна система. Наприклад, ураган в Атлантиці має можливість впливати на розподіл тиску у Європі. Інколи він може навіть пробитися до Європи як посттропічний шторм. Додаток Ventusky дозволяє проілюструвати взаємозалежність всієї системи, відображаючи розвиток тиску, вітру, хмарного покриву, опадів та температури на карті.

«The Ventusky» створили абсолютно нову систему відображення хвиль. За допомогою використання анімованих дуг візуалізація чітко розмежовує напрямок руху та висоту як вітрових хвиль, так і напрямків. Для інших метеорологічних елементів було обрано кольорові шкали, які належним чином ілюструють опади, тиск повітря та температуру. Синя температура показує прохолодну погоду, тоді як темно-червоний зображує гаряче пустельне повітря. Сині кольори представляють низьку кількість опадів, що не призведе до повеней. Помаранчеві та червоні відтінки, з іншого боку, небезпечні і можуть спричинити затоплення.

Додаток Ventusky доступний для всіх у всьому світі, спрямований на покращення поінформованості про метеорологічні події в нашій атмосфері (рис. 1.3).

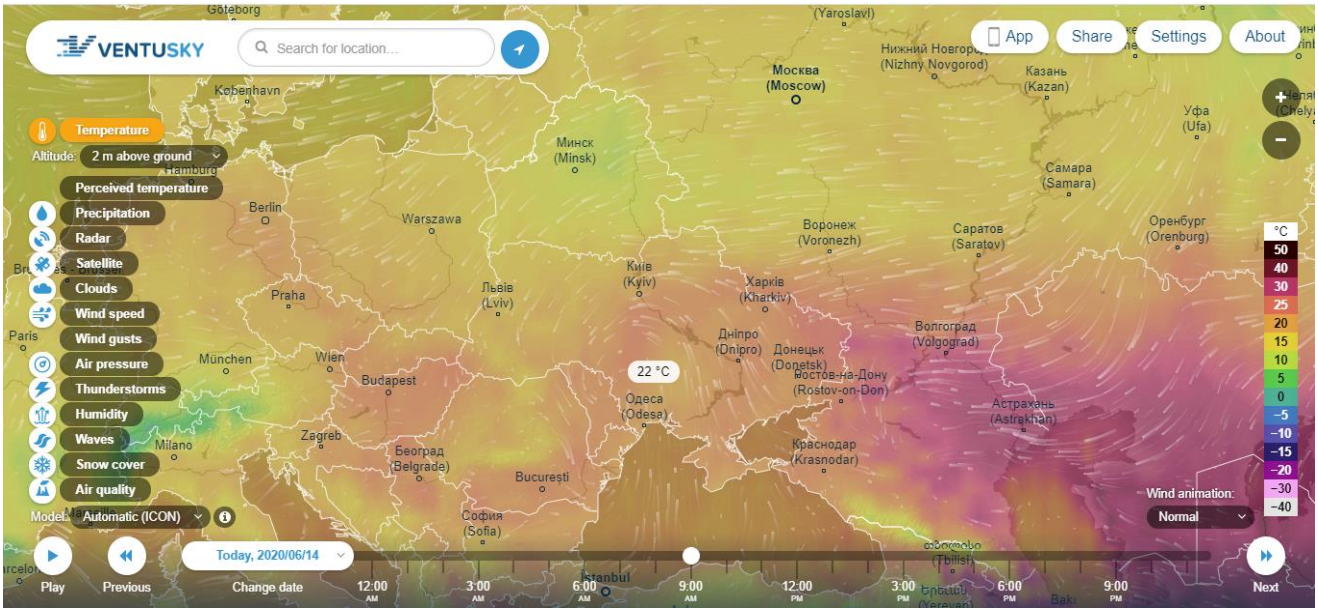

Рисунок 1.3 – Метеосервіс «The Ventusky»

1.2 Недоліки реалізації існуючих віртуальних музеїв.

Недоліками сучасних реалізацій метеосервісів є те, що в основу покладено незручний інтерфейс з інтегрованою рекламою в кожному вільному кутку. Що не дає змоги в повному обсязі передати інформацію про предмет та повністю дослідити його. Також сучасні метеосервіси дають невірні звіти погоди через те, що при розробці були використані старі технології, які не дають змогу інтегрувати сучасне API нових метеоцентрів.

1.3 Постановка задачі

За результатми проведеного аналізу недоліків існуючих віртуальних музеїв поставлені такі задачі:

1. вибрати середовище розробки для створення веб-додатку;
2. вибрати сучасні технології розробки веб-додатків;
3. вибрати метод створення веб-додатку;
4. поєднати вибрані методи рішення в одну інформаційну технологію;
5. розробити і протестувати розроблену інформаційну систему.

2 ВИБІР МЕТОДУ РІШЕННЯ

2.1 Вибір середовища розробки для створення веб-додатку.

Основним початковим єтапом кожної розробки веб-продукту являється вибір середовища розробки.

Для аналізу було підібрані найбільш популярні середовища розробки:

1. VS Code;
2. NetBeans;
3. PyCharm;
4. IntelliJ IDEA;
5. Eclipse;
6. Code:Blocks;
7. Aptana Studio 3;
8. Komodo;
9. RubyMine;
10. Xcode;

Вибір кожного з параметрів якості оцінювання базується на стандарті ISO 9126:2001, відповідно кожна з характеристик описується за допомогою кількох вхідних у неї атрибутів.

Для кожного з атрибутів визначається набір метрик, що дозволяють його оцінити. В нашому випадку, це: «Зручність використання», «Продуктивність» або «Часова ефективність», «Функціональність», «інтерфейс програми», «кількість підтримуємих бібліотек», а також параметр –«якість сбірки проекту» (кінцевого зображення після обробки).

Потрібно зазначити, що кожний з наведених критеріїв не є рівнозначним, тому умовно назначаються коефіцієнти важливості кожному з них (Таблиця 2.1).

Таблиця 2.1 Параметри оцінювання

Параметр	**Індекс**
Якість сбірки проекту	4,0
Кількість підтримуємих бібліотек	2,0
Інтерфейс	1,5
Зручність у використанні	2,0
Часова ефективність	2,5
Функціональність	3,0

Під час рейтингового оцінювання було виставлено оцінки від 1 до 10 (Таблиця 2.2), на основі досвіду роботи з подібними програмами, переглянутих відео, коментарях користувачів даного ПЗ, якості сбірки та інших факторах.

Таблиця 2.2 Таблиця оцінювання середовищ розробки

Назва ПЗ	Якість фінальної обробки	Кількість бібліотек	Зручність у використанні	Часова ефективність	Функціоналність	Інтерфейс
VS Code	9	10	10	8	9	10
NetBeans	8	6	7	8	9	8
PyCharm	4	5	8	7	4	4
IntelliJ IDEA	8	9	9	8	8	10
Eclipse	6	7	7	8	5	8
Code:Blocks	4	2	3	6	5	3
Aptana Studio 3	3	2	4	7	4	2
Komodo	7	4	6	5	8	5
RubyMine	4	7	4	6	8	5
Xcode	6	8	4	7	5	7

Кінцева рейтингова оцінка дорівнює сумі балів за кожний параметр, помножений на відповідний індекс оцінювання. Після обрахунків можна сформувати остаточну рейтингову таблицю (Таблиця 2.3).

Таблиця 2.3 Рейтингова таблиця ПЗ

Місце	Назва ПЗ	Оцінка
1	VS Code	137,5
2	IntelliJ IDEA	129,5
3	NetBeans	127
4	Eclipse	117,5
5	Xcode	114,5
6	RubyMine	106
7	PyCharm	96
8	Code:Blocks	84
9	Komodo	80
10	Aptana Studio 3	65

За результатами аналізу було вибрано програмне забезпечення для створення веб-додатку. Обрано VS Code - повнофункціональний професійний застосунок, середовище розробки, система для створення і редагування коду, розроблена компанією Microsoft. Містить найсучасніші засоби для розробників і фахівців в області програмування. Працює в операційних системах Microsoft Windows і Windows NT (як в 32-бітових, так і в 64-бітових).

VS Code використовується для створення веб-додатків, комп'ютерних ігор тощо. Також VS Code має свій власний зневаджувач в реальному часі - це дає змогу розробникам прискорити розробку за рахунок видимості змін при написанні коду. Широкий спектр інтегрованих розширеннь також дозволяє упростити та покращити швидкість розробки.

2.2 Вибір мови програмування для розробки веб-додатку.

Розробка веб-додатків набула великої популярності, а значить зростає набір інструментарію для їх створення. В даному розділі порівняємо (Таблиця 2.4) та оберемо інструмент для реалізації нашої мети.

Список популярних мов програмування для розробки веб-додатків:

1. PHP;
2. JavaScript;
3. Python.
4. Java
5. C
6. C++

Таблиця 2.4 Таблиця оцінювання мов програмування

Інструмент	**Плюси**	**Мінуси**
PHP	Орієнтація на Web-розробку; Кросплатформеність;	Непослідовний синтаксис; Старі процедурні артефакти
JavaScript	Корисні функціональні налаштування; Постійно підримується та вдосконалюється;	Відсутня типізація даних
Python	Широке застосування; Багато якісних бібліотек; Легкий в налаштуванні	Мала продуктивність та швидкість компіляції; Занадто монолітний;
Java	Розробка через тестування; Гнучкість; Багатопотоковість;	Платне комерційне використання; Низька продуктивність;
C	Продуктивність; Відносна простота мови;	Не використовується в сучасній веб-розробці;
C++	Висока продуктивність; Функціональне програмування;	Низькорівнева мова програмування; Складність знаходження помилок;

Проаналізувавши плюси та мінуси було обрано JavaScript так як ми закриваємо всі наші бажання функціоналом цієї мови.

JavaScript – динамічна скриптова мова програмування що має супутній інструментарій, призначений для розробки додатків у браузерах. Дозволяє користувачам створювати інтерактивний контент за допомогою наявних на ринку інструментів для веб-розробки.

2.3 Вибір фреймворку для створення веб-додатку.

Стандарти веб-розробки постійно зростають разом зі складністю сучасних технологій. За допомогою фреймворків складність розробки веб сервісів зменшується. В цьому розділі буде обрано фреймворк для розробки веб сервісу для відображення погоди.

Для аналізу було підібрані найбільш популярні фреймоврки для створення веб-сервісів:

1. Vue;
2. Angular;
3. React;
4. JQuery;
5. Node;
6. Titanium;

Під час рейтингового оцінювання фреймоврків були визнічені плюси та мінуси (Таблиця 2.5) на основі коментарях користувачів даного ПЗ

Таблиця 2.5 Таблиця оцінювання фрейморків

Фреймворк	**Плюси**	**Мінуси**
Vue	Легка інтеграція в проекти з використанням інших бібліотек; Створення SPA-додатків;	Компонентний підхід; Система рендеринга надає менше можливостей у порівнянні з іншими;
Angular	Підтримуються різні елементи MVC; Гнучкий; Пакети для розробки API	API Angular величезна, і потрібно розібратися з багатьма концепціями;

React	Free and Open Source; Швидкий розвиток; Підтримує віртуальну функціональність DOM;	Алгоритм Virtual DOM неточний і повільний; Потрібне складне асинхронне програмування при спілкуванні з сервером
JQuery	Широко використовується завдяки швидкій обробці; У всіх браузерах поводиться однаково	Безліч функцій, що полегшують роботу з DOM, вже реалізовані нативно; Може бути нестабільним
Node	Можливість застосовувати одну мову на клієнті і сервері; Технологія стрімко поліпшується	Треба постійно стежити за оновленнями, деякі речі виходять недостатньо протестованими;
Titanium	Простота навчання та реалізації; Високопродуктивна структура	Неефективний підхід до створення UI, на відміну від того ж React

На основі дослідження плюсів та мінусів популярних фреймворків для розробки веб сервісів було обрано Vue фреймворк - програмний каркас з відкритим кодом та контейнери з підтримкою інтеграції сторонніх бібліотек.

Vue - це веб-фреймворк призначений для розробки інтерфейсів на мові програмування JavaScript. Vue створений для поступового впровадження та росширення вже існуючих сервісів. Він вирішує різні завдання рівня уявлення (view), спрощує роботу з іншими бібліотеками та дозволяє створювати складні односторінкові додатки (SPA, Single-Page Applications).

Для роботи з фреймворком, вам вже потрібно знати HTML, CSS і звичайно ж JavaScript хоча б на базовому рівні.

Vue.js надає чудову можливість для реалізації сучасної архітектури MVC використовуючи шаблони та підключення компонентів до проекту.

Принцип MVC у програмуванні (Model-View-Controller, Модель-Подання-Контролер) - одна з найбільш вдалих ідей. На перший погляд принцип

MVC зрозумілий на інтуїтивному рівні, але не дуже простий якщо поглибитись в деталі (рис. 2.1) [3].

Такий підхід призводить до структурованого коду та дозволяє працювати над проектом більш спеціалізованим командам розробників, робить його більш зрозумілим і логічним, спрощує підтримку коду. Зміна в одному компоненті мінімально впливає на інші. До однієї моделі можна підключати різні контролери та різні види.

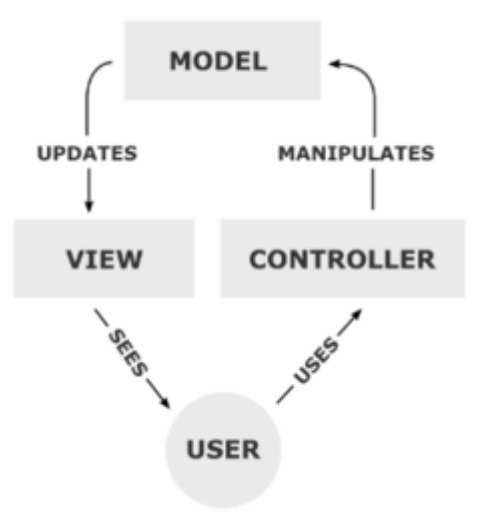

Рисунок 2.1 – Модель MVC

1. За допомогою Model ми інкапсулюємо дані додатка.
2. View відповідає за надання даних моделі та генерує DOM, який інтерпретується браузером клієнта.
3. Controller в цілому відповідає за обробку запитів і побудови відповідної моделі та передає його в представлення для рендеру.

Vue дозволяє розділяти весь код програми на компоненти і збирати їх в єдиний додаток. Компоненти можна використовувати повторно в будь-яких інших додатках.

Компонент - це екземпляр Vue з попередньо встановленими опціями показано на рисунку 2.2[4].

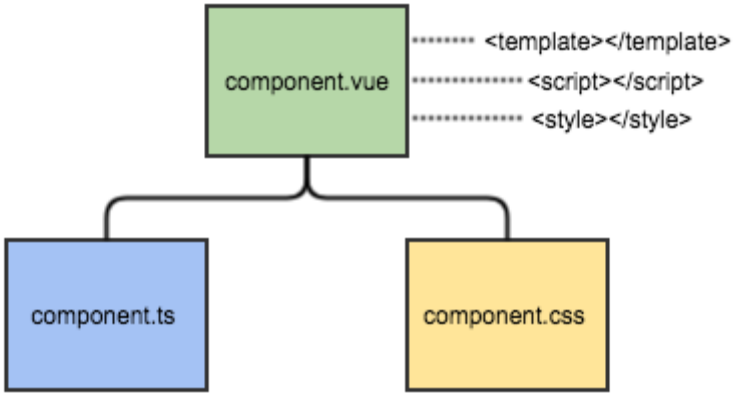

Рисунок 2.2 – Компонентна структурура vue.js

Для підключення сторонніх бібліотек будем використовувати пакетний менеджер NPM(Node Package Manager) показаний на рисунку 2.3[5].

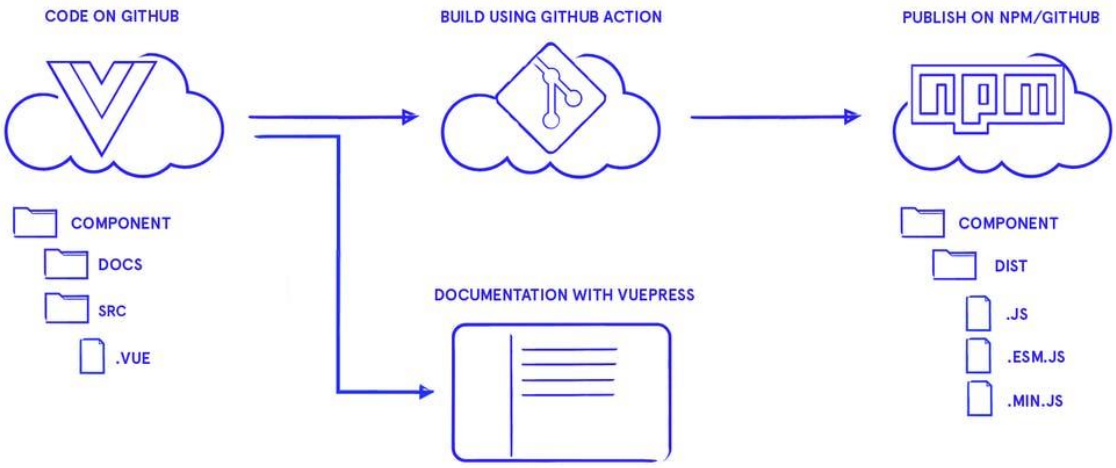

Рисунок 2.3 - NPM(Node Package Manager)

Наприклад, якщо ви хочете, щоб ваш проект використовував Bootstrap, вам необов'язково вручну копіювати всі необхідні файли з сайту Bootstrap і зберігати цю бібліотеку в репозиторії GIT, досить вказати в конфігураційному файлі вашого проекту одним рядком, що вам потрібен Bootstrap і пакетний менеджер npm встановить Bootstrap в копію вашого проекту, при цьому немає потрібності зберігати всі подібні бібліотеки в репозиторії GIT вашого проекту.

Це схоже на Maven для Java або Composer для PHP. Існує два основних інтерфейсу, з якими ви будете взаємодіяти: сайт NPM і набір інструментів командного рядка (CLI).

Зверніть увагу, що NPM поставляється разом з бінарним файлом Node.js, тому вам не потрібно його встановлювати, якщо ви маэте встановлений node.js, проте якщо ви хочете використовувати специфічну версію NPM, ви можете його відновити.

2.4 Поєднання вибраних методів рішення в одну інформаційну технологію.

Проаналізувавши та вибравши компоненти для проектування метеосервісу було створено наступну технологічну схему роботи веб-додатку, що відображено на рисунку 2.4.

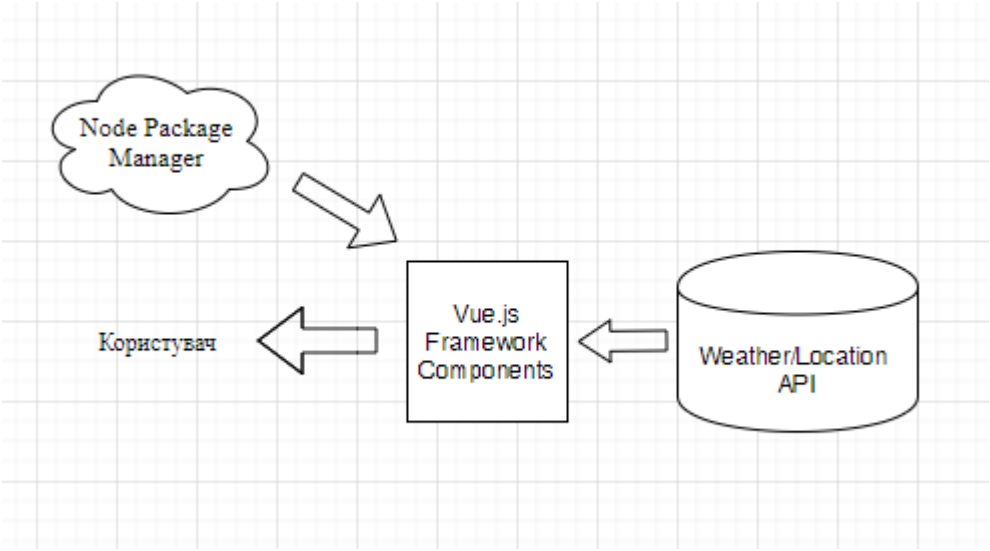

Рисунок 2.4 – Інформаційна технологія для метеосервісу

3 ІНФОРМАЦІЙНЕ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ СИСТЕМИ

3.1 Інформаційна модель метеосервісу та основні діаграми веб-додатку.

Робота над веб-додатком починається зі створення діаграми послідовності, діаграма варіантів використання, діаграми класів. Створення діаграми представлені в цьому підрозділі.

Діаграма послідовності (англ. Sequence diagram) - діаграма, на якій для деякого набору об'єктів на єдиній тимчасової осі показаний життєвий цикл будь-якого певного об'єкта (створення-діяльність-знищення якоїсь сутності) і взаємодія акторів (дійових осіб) ІС в рамках якого- або певного прецеденту (відправка запитів і відповідей) отримання.

Діаграма послідовнестой веб-додатку для створення метеосервісу відображена на рисунку 3.1.

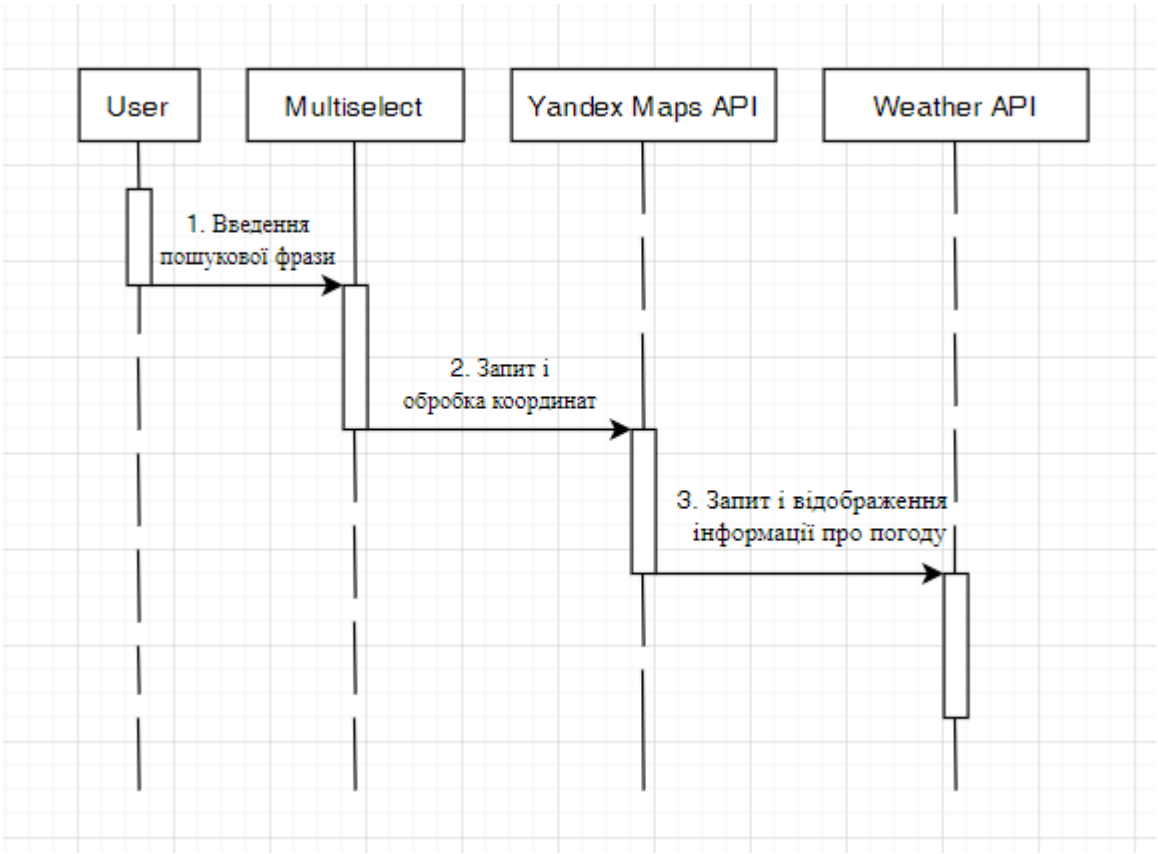

Рисунок 3.1 – Діаграма послідовнестой веб-додатку

Діаграма варіантів використання (англ. Use case diagram) в UML - діаграма, що відображає відносини між акторами і прецедентами і є складовою частиною моделі прецедентів, що дозволяє описати систему на концептуальному рівні.

Прецедент - можливість модельованої системи (частина її функціональності), завдяки якій користувач може отримати конкретний, вимірний і потрібний йому результат. Прецедент відповідає окремому сервісу системи, визначає один з варіантів її використання і описує типовий спосіб взаємодії користувача з системою. Варіанти використання зазвичай застосовуються для специфікації зовнішніх вимог до системи.

Діаграми варіантів використання застосовуються при бізнес-аналізі для моделювання видів робіт, виконуваних організацією, і для моделювання функціональних вимог до ПС при її проектуванні і розробці. Побудова моделі вимог при необхідності доповнюється їх текстовим описом. При цьому ієрархічна організація вимог представляється за допомогою пакетів use cases.

На рисунку 3.2 представлена спроектована діаграма для веб додатку.

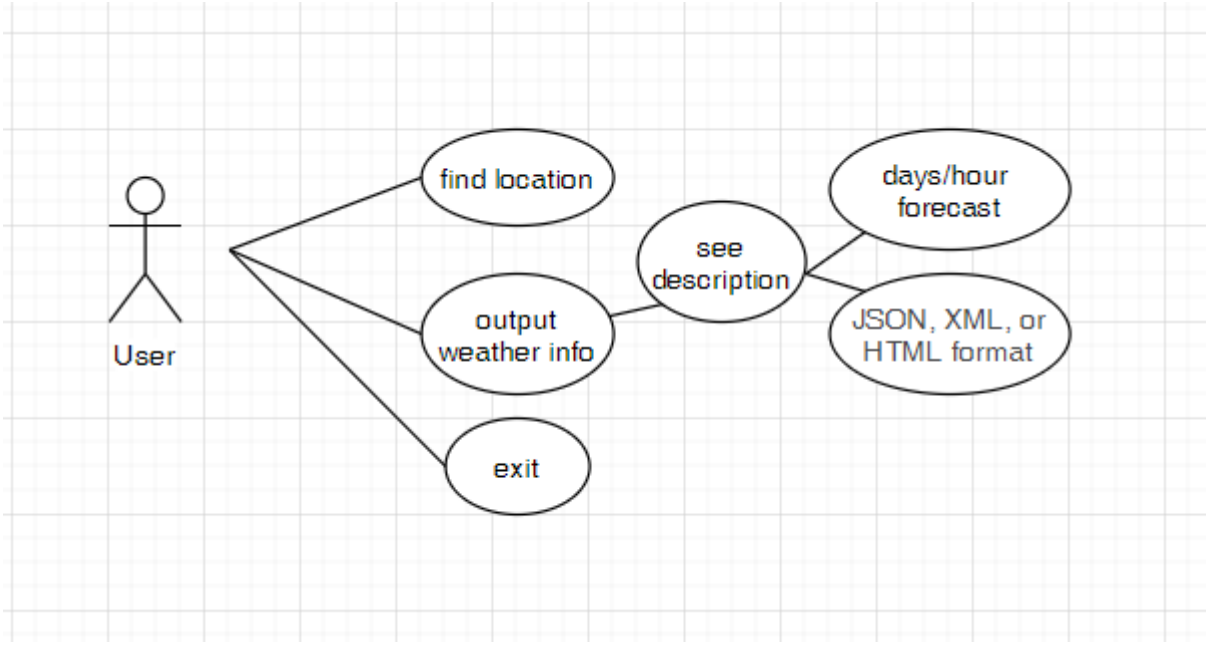

Рисунок 3.2 – Діаграми варіантів використання веб-додатку

ER-модель (від англ. Entity-relationship model, модель «сутність - зв'язок») - модель даних, що дозволяє описувати концептуальні схеми предметної області.

ER-модель використовується при високорівневої (концептуальному) проектуванні баз даних. З її допомогою можна виділити ключові сутності і позначити зв'язки, які можуть встановлюватися між цими сутностями.

Під час проектування баз даних відбувається перетворення ER-моделі в конкретну схему бази даних на основі обраної моделі даних (реляційної, об'єктної, мережевий або ін.).

ER-модель являє собою формальну конструкцію, яка сама по собі не наказує ніяких графічних засобів її візуалізації. Як стандартна графічної нотації, за допомогою якої можна візуалізувати ER-модель, була запропонована діаграма "сутність-зв'язок» (англ. Entity-relationship diagram, ERD, ER-діаграма).

Поняття «ER-модель» і «ER-діаграма» часто вже не розрізняють, хоча для візуалізації ER-моделей можуть бути використані і інші графічні нотації, або візуалізація може взагалі не застосовуватися (наприклад, використовуватися текстовий опис).

Модель була запропонована в 1976 році Пітером Ченом, їм же запропонована і найпопулярніша графічна нотація для моделі.

Розроблена ER діаграма веб додатку представлена на рисунку 3.3. Створенні сутності Product – містить інформацію про продукт в музеї. Category – інформація про категроію продукту. Users – зберігає інформацію стосовно користувача. Authorities – якмими правами наділений користувач веб додатку.

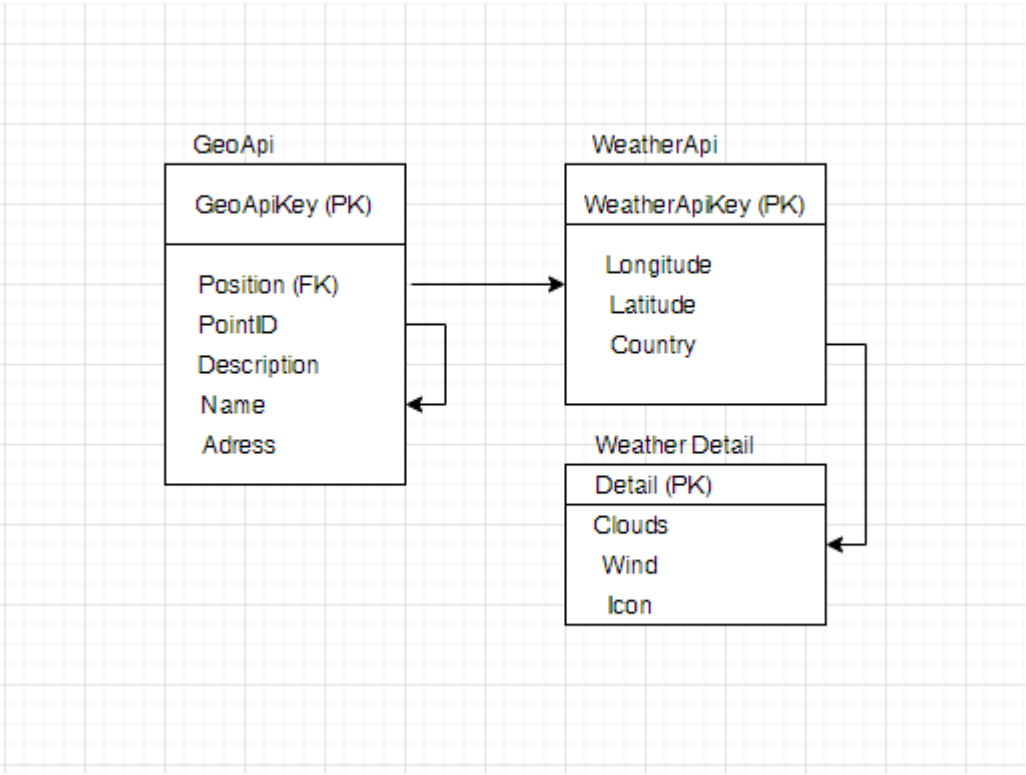

Рисунок 3.3 - ER-діаграма метеосервісу

Розробимо діаграму класів майбутнього веб додатку для реалізації метеосервісу.

Діаграма класів (англ. Static Structure diagram) - структурна діаграма мови моделювання UML, що демонструє загальну структуру ієрархії класів системи, їх кооперацій, атрибутів (полів), методів, інтерфейсів і взаємозв'язків між ними. Широко застосовується не тільки для документування та візуалізації, але також для конструювання за допомогою прямого або зворотного проектування.

Метою створення діаграми класів є графічне представлення статичної структури декларативних елементів системи (класів, типів і т.п.) Вона містить в собі також деякі елементи поведінки (наприклад - операції), проте їх динаміка повинна бути відображена на діаграмах інших видів (діаграмах комунікації, діаграмах станів). Для зручності сприйняття діаграму класів можна також доповнити поданням пакетів, включаючи вкладені.

При поданні сутностей реального світу розробнику потрібно відобразити їх поточний стан, їх поведінку і їх взаємні відносини. На кожному етапі здійснюється абстрагування від незначних деталей і концепцій, які не належать

до реальності (продуктивність, інкапсуляція, видимість і т.п.). Класи можна розглядати з позиції різних рівнів. Як правило, їх виділяють три основних: аналітичний рівень, рівень проектування і рівень реалізації :

- на рівні аналізу клас містить у собі тільки начерк загальних контурів системи і працює як логічна концепція предметної області або програмного продукту.
- на рівні проектування клас відображає основні проектні рішення щодо розподілу інформації і планованої функціональності, об'єднуючи в собі відомості про стан та операції.
- на рівні реалізації клас допрацьовується до такого виду, в якому він максимально зручний для втілення в вибраному середовищі розробки; при цьому не забороняється опустити в ньому ті загальні властивості, які не застосовуються на обраною мовою програмування.

Можна скільки завгодно сперечатися, який фреймворк краще, але не можна не визнати очевидне - вони всі базуються на компонентах. У React, в Vue, в Angular ви займаєтеся тим, що ділите своє додаток на невеликі частини і працюєте з ними як з самостійними одиницями.

Концепція компонентного підходу у фронтенді відкриває неймовірні можливості для повторного використання написаного коду. Тільки уявіть, ви створюєте компонент (наприклад прелоадер) для одного проекту, а потім використовуєте його у всіх інших, без всякого переписування або рефакторингу.

Діаграма компонентів представлена на рисунку 3.4. На ній можна виділити:

1. Компоненти, що описують моделі додатку;
2. Сервісний шар додатку;
3. Компоненти контролери;
4. За представлення відповідають VUE файли.

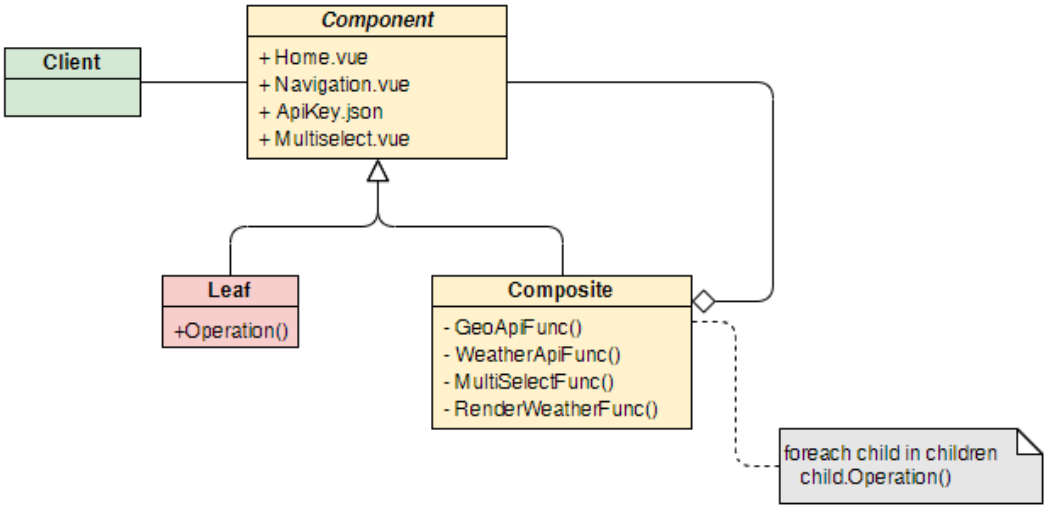

Рисунок 3.4 – Діаграма компонентів

3.2 Реалізація веб-додатку.

На основі створених діаграм розробимо веб додаток для функціонування метеосервісу. Структура проекту веб додатку зображена на рисунку 3.5.

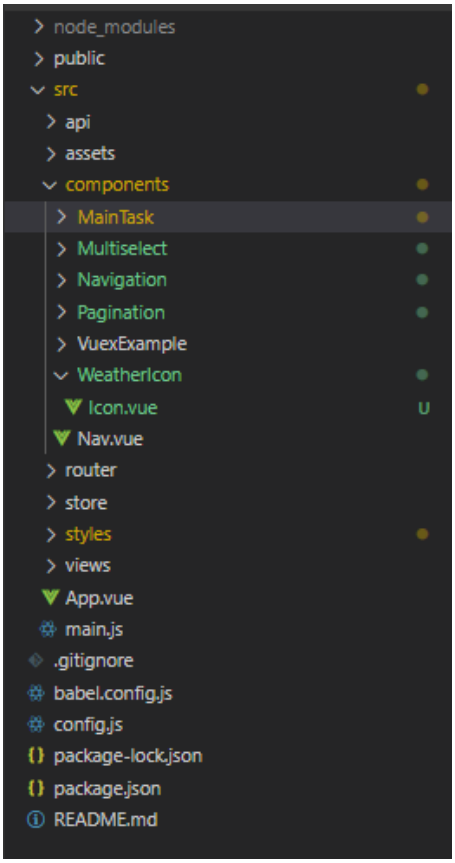

Рисунок 3.5 – Структура проекту веб додатку

Проект веб сервісу складається з 5-ти основних папок:

1. node_modules – містить головні компоненти фреймоврку Vue;
2. components – містить компоненти та модулі нашого веб-додатку;
3. views – містить інформацію для відображення веб сторінок та файли конфігурації для запуску веб додатку;
4. router – бібліотека маршрутизації;
5. store – контейнер станів веб-додатку.

В сучасних проектах переважно використовують системи збірки. Для даного веб додатку буде задіяний NPM(Node Package Manager).

Перед початком написання коду потрібно підключити необхідні бібліотеки. Нижче наведено необхідний набір бібліотек для проекту Axios, Bootstrap, moment, vue-carousel, multiselect, vue-router, Vuex, EsLint.

```
{
  "name": "WeatherService",
  "version": "0.1.0",
  "private": true,
  "scripts": {
    "serve": "vue-cli-service serve",
    "build": "vue-cli-service build",
    "lint": "vue-cli-service lint"
  },
  "dependencies": {
    "axios": "^0.19.0",
    "bootstrap": "^4.4.1",
    "bootstrap-vue": "^2.1.0",
    "core-js": "^3.4.3",
    "es6-promise": "^4.2.8",
    "moment": "^2.24.0",
    "semantic-ui-card": "^2.3.1",
    "vue": "^2.6.10",
    "vue-carousel": "^0.18.0",
    "vue-moment": "^4.1.0",
    "vue-multiselect": "^2.1.6",
    "vue-router": "^3.1.3",
    "vuex": "^3.1.2"
  },
  "devDependencies": {
    "@vue/cli-plugin-babel": "^4.1.0",
    "@vue/cli-plugin-eslint": "^4.1.0",
    "@vue/cli-plugin-router": "^4.1.0",
    "@vue/cli-service": "^4.1.0",
    "babel-eslint": "^10.0.3",
    "eslint": "^5.16.0",
    "eslint-plugin-vue": "^5.0.0",
    "node-sass": "^4.12.0",
    "sass-loader": "^8.0.0",
    "vue-template-compiler": "^2.6.10"
  },
```

```
  "eslintConfig": {
    "root": true,
    "env": {
      "node": true
    },
    "extends": [
      "plugin:vue/essential",
      "eslint:recommended"
    ],
    "rules": {},
    "parserOptions": {
      "parser": "babel-eslint"
    }
  },
  "browserslist": [
    "> 1%",
    "last 2 versions"
  ]
}
```

Для того, щоб Axios міг обробляти HTTP запити, потрібно прописати відповідні конфігураційні налаштування в файлі router.js:

```
import Vue from 'vue'
import VueRouter from 'vue-router'
import Home from '../views/MainTask.vue'
import VuexExample from '@/views/AppVuex.vue'
import Chapter1 from '@/views/Multiselect.vue'
import Chapter2 from '@/views/Navigation.vue'
import Chapter3 from '@/views/Pagination.vue'
import Navigation from '@/views/WeatherIcon.vue'  // ---> тогда в пути
вместо import вставляю 'Navigation'

Vue.use(VueRouter)

const routes = [
  {
    path: '/',
    name: 'WeatherServise',
    component: Home
  },
  {
    path: '/vuex',
    name: 'VuexExample',
    component: VuexExample
  },
  {
    path: '/Multiselect,
    name: Multiselect,
    component: Multiselect
  },
  {
    path: '/Pagination,
    name: Pagination,
    component: Pagination
  },
  {
    path: '/WeatherIcon,
    name: WeatherIcon,
    component: WeatherIcon
  },
```

```
  {
    path: '/Navigation',
    name: 'Navigation',
    component: Navigation
  },
  {
  path: '/weather',
  name: 'weather',
  component: Home
  }
]

const router = new VueRouter({
  mode: 'history',
  base: process.env.BASE_URL,
  routes
})

export default router
```

Після ініціалізації контейнеру станів Vuex потрібно загрузити контекст. Потрібно розбити контейнер на модулі для кожного логічного стану компонентів та написати для кожного модуля слухачі Getters, Actions та Mutations, як показано нижче:

```
import shop from '../../api/weather'

const state = {
  items: [],
  checkoutStatus: null
}

// getters
const getters = {
  searchStates: (state, getters, rootState) => {
    return state.items.map(({ id, quantity }) => {
      const product = rootState.products.all.find(product => state.id ===
id)
      return {
        title: search.title,
        coordinate: search.coordinate
      }
    })
  },

  totalAim: (state, getters) => {
    return getters.totalAim.reduce((total, search) => {
      return total + search.name * search.adress
    }, 0)
  }
}

// actions
const actions = {
  checkout ({ commit, state }, search) {
    const savedAdress = [...state.items]
    commit('setCheckoutStatus', null)
    commit('setLon', { items: [] })
    weather.api(
```

```
      search,
      () => commit('setCheckoutStatus', 'successful'),
      () => {
        commit('setCheckoutStatus', 'failed')
        // rollback to the api saved before sending the request
        commit('setLonAndLat', { items: savedSearch })
      }
    )
  },

  sendCoordinate ({ state, commit }, coordinate) {
    commit('setCheckoutStatus', null)
    if (item.inventory > 0) {
      const container = state.items.find(item => item.id === search.id)
      if (!coordinate) {
        commit('pushCoordinate ',
    { id: coordinate.id })
      } else {
        commit('reset', item)
      }
      // remove 1 item from stock
      commit('products/'pushCoordinate, { id: item.id }, { root: true })
    }
  }
}

// mutations
const mutations = {
  pushCoordinate (state, { id }) {
    state.items.push({
      id,
    })
  },

  resetCoordinate (state, { id }) {
    const coordinate = state.items.find(item => item.id === id)
    coordinate.item = []
  },

  setCartCoord (state, { items }) {
    state.items = items
  },

  setCheckoutStatus (state, status) {
    state.checkoutStatus = status
  },

  resetAll (state, status) {
    state = ...state
  }
}

export default {
  namespaced:
  true,
  state,
  getters,
  actions,
  mutations
}
```

В теці styles містяться конфігураційні файли для налаштування стилів нашого сервісу (рис. 3.6).

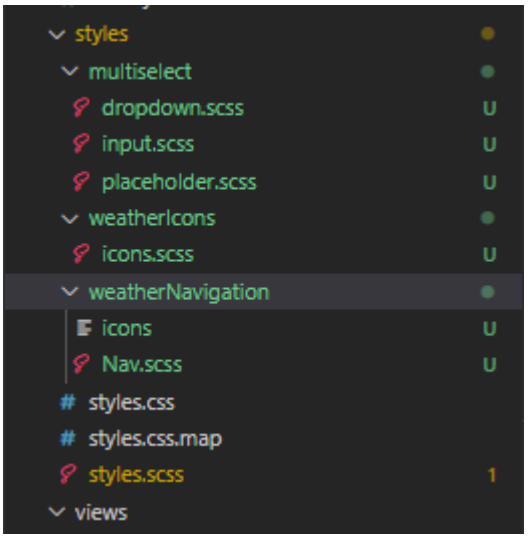

Рисунок 3.6 – Структура стилів веб додатку

Після налаштування стилів переходимо до створення компонентів веб-додатку. На рисунку 3.7 показана структура папки components. Можна виділити такі папки як Multiselect, Navigation, Pagination, Weather.

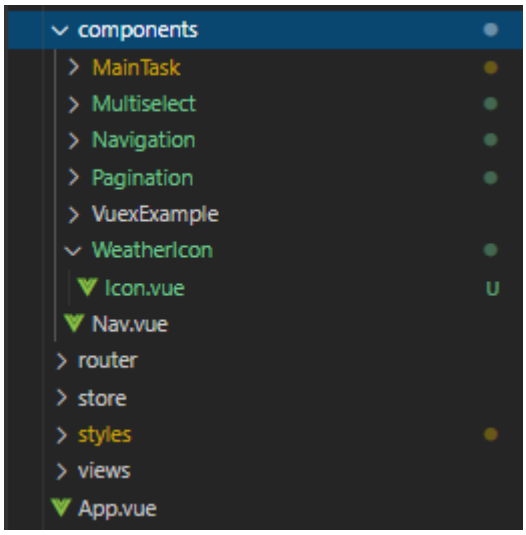

Рисунок 3.7 – Структура компонентів веб-додатку

Створюємо головний керуючий компонент на основі нашої ER діаграми (рис. 3.3). Кожен з методів описує об’єкти веб додатку.

Приклад двух головних методів, які звертаючись до гео-API, обмінюються параметрами та зберігають інформацію звіту погоди, яку користувач хоче побачити:

```
getGeoApi(currentValue) {
```

```
   this.temp = [];
   this.time = [];
if (currentValue.length <= 2) {
   this.options = [];
   this.temp = [];
} else {
this.$axios.get(`${config.geoApi}?apikey=${config.apiKeyGeo}&format=json&geo
code=${currentValue}`)
   .then( response => {
   console.log(response)
   this.geo = response.data.response.GeoObjectCollection.featureMember;

   this.geo.forEach(obj => {
      this.options.push(obj.GeoObject); // вытаскиваю все объекты с массива
api
      this.coord.push(obj.GeoObject.Point.pos); // вытаскиваю координаты
   });
   console.log(this.options)
  })
 }
},

axiosWeatherApi() {
  var coord = currentValue.Point.pos.split(" ");
  this.lon = coord[0];
  this.lat = coord[1];
  this.axiosWeatherApi();
  this.$axios.get(`${config.weatherApi}lat=${this.lat}&lon=
  ${this.lon}&units=metric&appid=${config.apiKeyWeather}`)
    .then( response => {
       console.log(response)
       this.weather = response.data.list;
        this.weather.forEach(obj => {
           this.temp.push(obj.main.temp);
           this.time.push(obj.dt_txt);
        });
     this.displayWeather(this.weather);
   })
}
```

За допомогою методу displayWeather() відбуваєть рендер інтерфейсу перегляду звіту погоди:

```
displayWeather(value) {
   let sliceIndex = 0;
   let sliceEnd;
   if (value && value.length) {
     this.errorWeatherData = false;
     this.weatherData = [];
     let currentDay = moment.unix(value[0].dt).utc().get('date'); // utc() -
fix timezone

     value.forEach( (item, index) => {
       let itemDay = moment.unix(item.dt).utc().get('date'); // utc() - fix
timezone
       sliceEnd = index;
       if (itemDay != currentDay) {
         this.weatherData.push(value.slice(sliceIndex, sliceEnd));
         currentDay = itemDay;
         sliceIndex = index;
       }
```

```
        });
        this.weatherData.push(value.slice(sliceIndex));
      } else {
        this.errorWeatherData = true;
        this.weatherData = [];
      }

      this.selectedValue = this.selected.name
      this.$router.push({ query: {
        name: this.selected.name,
        lat: this.lat, lon: this.lon,
        page: this.slide
      }})
      .catch(err => {console.log(err)})
  }
```

3.3 Етапи використання метеосервісу та тестування.

Робота з додатком в головному починається з пошуку місця в котрому користувач хоче знати погодні умови. Пошук міста чи країни (рис. 3.9) відбувається в мультиселект.

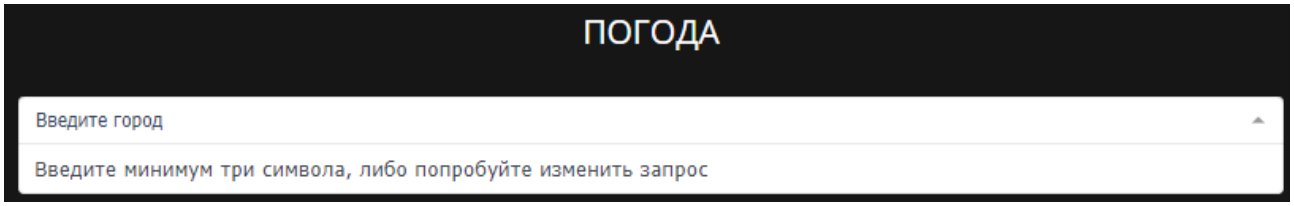

Рисунок 3.9 – Пошук місця, у якому користувач
бажає дізнатись погоду

Після того, як користувач почав вводити данні у поле запиту – веб-додаток робить запит до гео-API, та показує весь можливий список регіонів, які задовольняють умови запиту (рис. 3.10) .

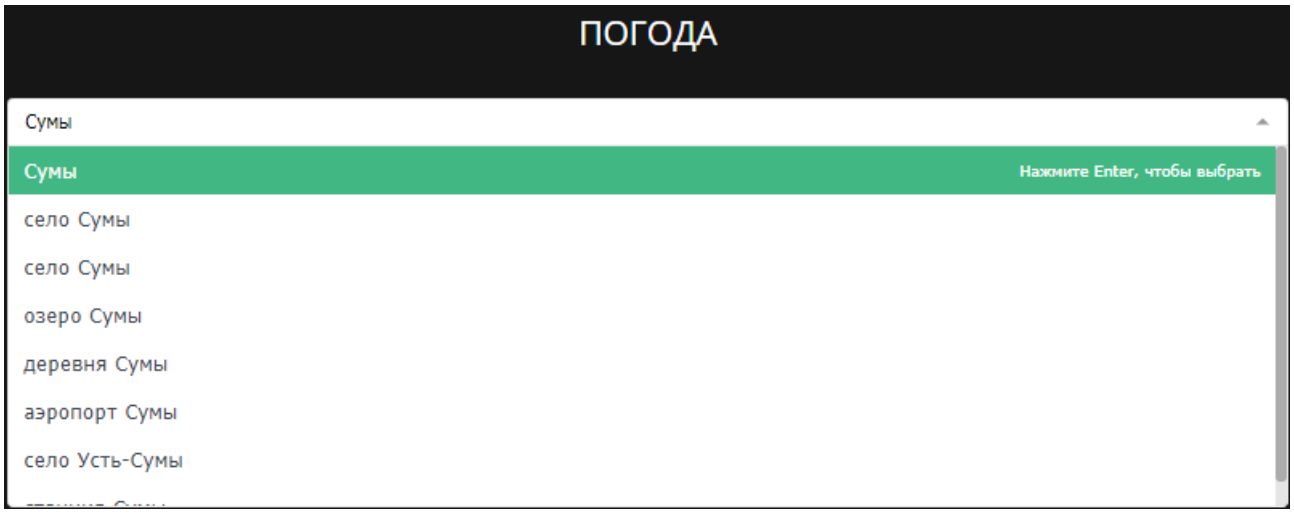

Рисунок 3.10 – Введення запиту в поле

Наступним кроком є виведення інформації по погоді в інтерфейс. У користувача є можливість перегляду даних звіту погоди у погодинному форматі, кожні 3 години в 5-ти наступних дні. Знизу знаходиться панель пагінації – переходу до наступного дня. Також при успішному запиті в url браузера зберігаються параметри Name – назва міста, Lat/Lon – координати довготи та широти, Page – відкрита сторінка. Відображення головної інформації на риунку 3.11 відбувається на самописному слайдері.

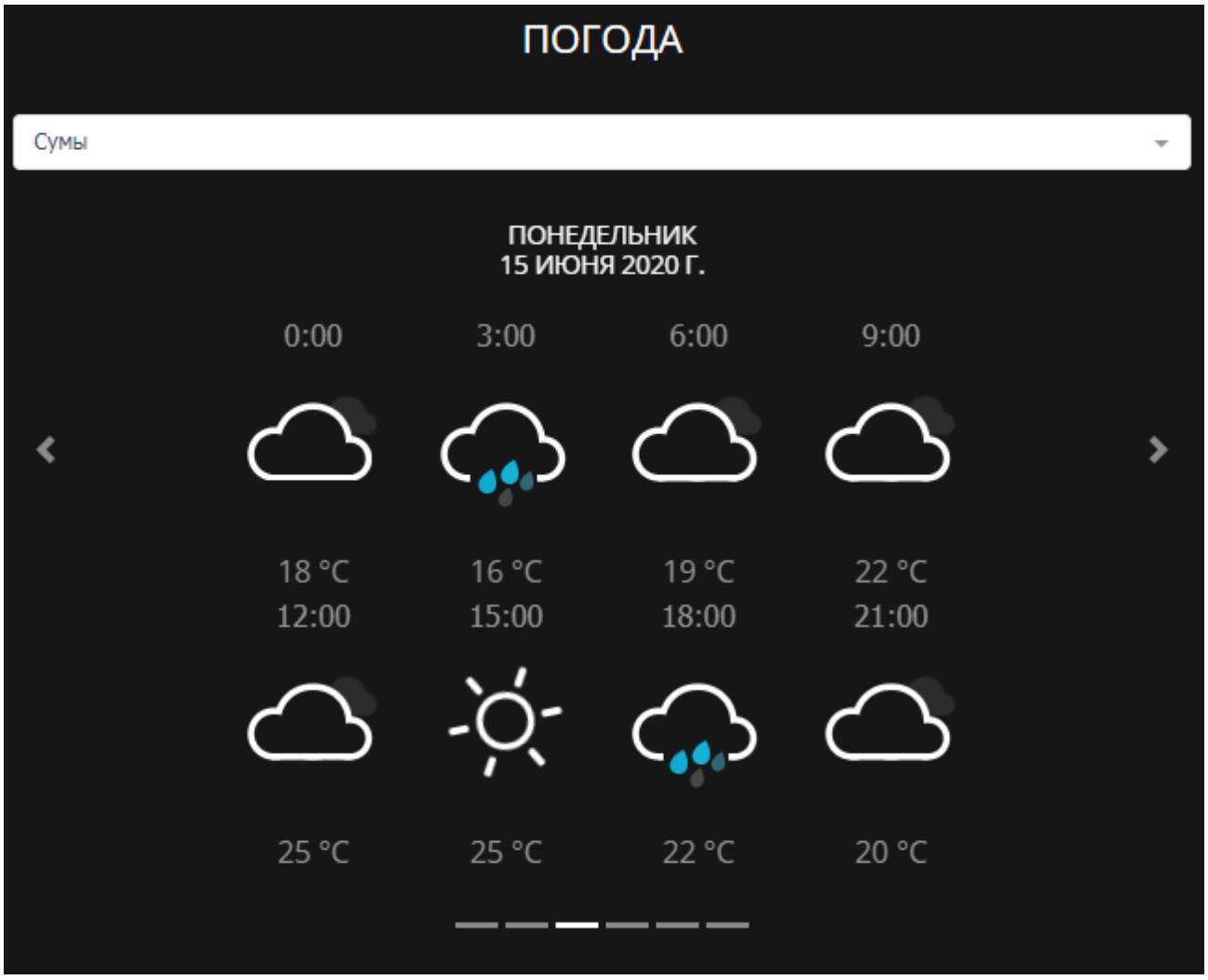

Рисунок 3.11 – Відображення звіту погоди

Перевіривши функціонування основних елементів веб додатку можна зробити висновок, що система працює нормально. Відображення інформації про погоду та самі запити до API метеостанцій здійсюються корректно. Система валідації та захисту працює в визначених межах.

ВИСНОВКИ

В ході виконная дипломної роботи було:

1. Проаналізовано відображення сучасного метеосервісу та було виявлено їх недоліки:

2. Обране середовище розробки для створення веб-сервісу. Після аналізу недоліків та переваг сучасних програм для реалізації метеосервісу вибір впав на програму VS Code, що дозволяє створювати кросплатформенні та облачні застосунки.

3. Обрана мова програмування JavaScript у поєднанні з фреймворком Vue.js. Використовуючи сучасні рішення від Vue можна досягти продуктивності та швидкості веб-додатку.

4. Для покращення обміну інструментами, установки різних модулів і управління їх залежностями був обраний пакетний метод – Node Package Manager.

5. Поєднано вибрані методи рішення в одну інформаційну технологію. Створено технологію роботи інформаційної системи. Розробленні схеми та діаграми для веб додатку.

6. Було реалізовано програмне забезпечення для функціонування метеосервісу, а також протестувано розроблену інформаційну систему.

Поєднавши разом інструменти створювання веб додатків було отримано інформаційну технологію - метеосервіс, що дозволяє якісно представити звітну інформацію по погоді користувачеві.

СПИСОК ЛІТЕРАТУРИ

1. Метеосервіс «Гісметео» - https://www.gismeteo.ru/
2. Метеосервіс «Ventusky» - https://www.ventusky.com/about
3. Model-View-Controller - https://ru.wikipedia.org/wiki/Model-View-Controller
4. Работа с данными на границе Vue.js-приложения. Постановка задач – https://habr.com/ru/company/ruvds/blog/505756/
5. Руководство по Node.js, часть 1: общие сведения и начало работы – https://habr.com/ru/company/ruvds/blog/422893/
6. Крокфорд. Д. Как устроен JavaScript: Учебное пособие / Д. Крокфорд, - М.: Миннесота, 2019. – 304 с.
7. Браун Є. Learning JavaScript: JavaScript Essentials for Modern Application Development / Є. Браун, Коваленко В.А.(переводчик), - М.: Бином, 2017. – 368 с.
8. Резиг Д. Секреты JavaScript / Резиг Д., Марас И., Бибо Б., - М.: Вильямс, 2017. – 544 с.
9. От нуля до деплоя: разработка системы документации с помощью Vue и VuePress – https://medium.com/devschacht/от-нуля-до-деплоя-разработка-системы-документации-с-помощью-vue-и-vuepress-cf6bde7c9a1f
10. Упрощаем работу с npm: полезные сокращения и трюки для разработки – https://tproger.ru/translations/npm-tricks/
11. Введение в пакетный менеджер NPM для начинающих.(A Beginner’s Guide) – http://prgssr.ru/development/vvedenie-v-paketnyj-menedzher-npm-dlya-nachinayushih.html
12. Vue: как использовать компоненты – https://medium.com/@mo-dex13/vue-js-2-как-использовать-компоненты-8f029ba5a60c
13. Используем Axios для доступа к API – https://ru.vuejs.org/v2/cook-book/using-axios-to-consume-apis.html

14. Модульное тестирование Vue-компонентов и файловая структура – https://ru.vuejs.org/v2/cookbook/unit-testing-vue-components.html

15. Совместимость с браузерами – https://cli.vuejs.org/ru/guide/browser-compatibility.html#browserslist

16. Лучшие практики JavaScript. Производительность, переменные – https://nuancesprog.ru/p/6829/

17. Composition API во Vue 3 — плюсы, минусы и опыт использования – https://tproger.ru/video/composition-api-in-vue/?autoplay=1

18. Компоненты всему голова: тренды компонентного подхода в 2020 – https://proglib.io/p/components-in-web

19. Стандартные директивы в Vue.js – https://monsterlessons.com/project/lessons/standartnye-direktivy-v-vuejs

ДОДАТОК

WeatherMain.vue

```
<template>
    <div class="weather">
        <h1>ПОГОДА</h1>
        <multiselect
        placeholder="Введите город"
        selectLabel="Нажмите Enter, чтобы выбрать"
        v-model="selected"
        :value='selectedValue'
        @select="getWeatherApi"
        @keyup="totalcharacter++"
        @search-change="getGeoApi"
        :options="options"
        label="name"
        track-by="name">
        <template slot="noResult">
            <span>Введите минимум три символа, либо попробуйте изменить
запрос</span>
        </template>
        <template slot="noOptions">
            <span>Введите минимум три символа, либо попробуйте изменить
запрос</span>
        </template>
        </multiselect>

        <b-carousel v-if="weatherData.length"
        id="carousel-1"
        ref="myCarousel"
        v-model="slide"
        :interval="0"
        controls
        indicators
        @sliding-end="onSlideEnd">

        <b-carousel-slide        v-for="weatherDay        in        weatherData"
:key="weatherDay.id">
          <h1 class="day">{{ moment(weatherDay[0].dt_txt).format('dddd') }} <br>
{{ moment(weatherDay[0].dt_txt).format('LL') }}</h1>
                <div class="example-slide">
                    <div v-for="weatherHour in weatherDay" :key="weatherHour.id"
class="slide-item">
                        <div                                class="weather-time">{{
moment.unix(weatherHour.dt).utc().format('LT') }}</div>
                        <!-- {{weatherHour.weather[0].main}} -->
                        <Icon :iconType="weatherHour.weather[0].main" />

                        <div                                class="weather-temp">{{
Math.round(weatherHour.main.temp) + ' °' + 'C' }}</div>
                    </div>
                </div>
      </b-carousel-slide>
    </b-carousel>

    </div>
</template>

<script>
/* eslint-disable no-console */
import Multiselect from 'vue-multiselect'
```

```
import moment from 'moment'
import config from '../../../config.js'
import Icon from './Icon'
export default {
    name: 'weather',
    components: { Multiselect, Icon },
    data () {
        return {
            totalcharacter : 0,
            selected: {},
            options: [],
            geo: {},
            weather: {},
            dateTime: {},
            coord: [],
            temp: [],
            time: [],
            weatherData: [],
            itemPerPage: 0,
            days: [],
            nextLabel: "<div class='nav-carousel-next'></div>",
            prevLabel: "<div class='nav-carousel-prev'></div>",
            lat: 0,
            lon: 0,
            page: 0,
            saveUrl: [],
            selectedValue: '',
            currentPage: 0,
            ccc: 0,

            slide: 0,
            sliding: null
        }
    },
    methods: {
        onSlideEnd() {
            this.sliding = false
            console.log(this.slide);
            this.$router.push({   query:   {   name:   this.selected.name,   lat:
this.lat, lon: this.lon, page: this.slide } }).catch(err => {console.log(err)})
        },
        getGeoApi(currentValue) {
        this.temp = [];
        this.time = [];
        if(currentValue.length <= 2) {
            this.options = [];
            this.temp = [];
        }else{

this.$axios.get(`${config.geoApi}?apikey=${config.apiKeyGeo}&format=json&geocode
=${currentValue}`)
            .then( response => {

console.log(`${config.geoApi}?apikey=${config.apiKeyGeo}&format=json&geocode=${c
urrentValue}`);

            this.geo = response.data.response.GeoObjectCollection.featureMember;

            this.geo.forEach(obj => {

                this.options.push(obj.GeoObject);  // вытаскиваю все объекты с
массива api
```

```
                this.coord.push(obj.GeoObject.Point.pos);      //      вытаскиваю
координаты

            });
            console.log(this.options)

            })
        }

        },
        getWeatherApi(currentValue) {

            var coord = currentValue.Point.pos.split(" ");
            this.lon = coord[0];
            this.lat = coord[1];

            this.axiosWeatherApi();

        },
        axiosWeatherApi() {

this.$axios.get(`${config.weatherApi}lat=${this.lat}&lon=${this.lon}&units=metri
c&appid=${config.apiKeyWeather}`)
            .then( response => {

console.log(`${config.weatherApi}lat=${this.lat}&lon=${this.lon}&units=metric&ap
pid=${config.apiKeyWeather}`);

            this.weather = response.data.list;

            this.weather.forEach(obj => {

                this.temp.push(obj.main.temp); // температура в Кельвинах за 5
дней (каждые 3 часа)
                this.time.push(obj.dt_txt); // дата и время (5 дней каждые 3
часа - 40 элементов по 8 элементов на целый день)

            });

            this.displayWeather(this.weather);

            })
        },
        displayWeather(value) {
            let sliceIndex = 0;
            let sliceEnd;
            if (value && value.length) {
                this.errorWeatherData = false;
                this.weatherData = [];
                let currentDay = moment.unix(value[0].dt).utc().get('date'); //
utc() - fix timezone

               value.forEach( (item, index) => {
                    let  itemDay  =  moment.unix(item.dt).utc().get('date');  //
utc() - fix timezone
                    // console.log(index);
                    sliceEnd = index;
                    if (itemDay != currentDay) {
                        this.weatherData.push(value.slice(sliceIndex,
sliceEnd));
                        currentDay = itemDay;
                        sliceIndex = index;
                    }
```

```
                });
                this.weatherData.push(value.slice(sliceIndex));
            } else {

                this.errorWeatherData = true;
                this.weatherData = [];
            }

            this.selectedValue = this.selected.name
            this.$router.push({   query:   {   name:   this.selected.name,   lat:
this.lat, lon: this.lon, page: this.slide } }).catch(err => {console.log(err)})

        }
    },
    created() {
        if( this.$route.query.lat && this.$route.query.lon ) {

            this.selected.name = this.$route.query.name;
            this.lat = this.$route.query.lat;
            this.lon = this.$route.query.lon;

            this.selectedValue = this.$route.query.name; // value multiselect
            this.slide = Number(this.$route.query.page); // current slide

            console.log(this.currentPage)

            this.axiosWeatherApi();

        }
        console.log(this.weatherHour, 666);

    }
}
</script>

<!-- Add "scoped" attribute to limit CSS to this component only -->
<style lang="scss">
// @import "~vue-multiselect/dist/vue-multiselect.min.css";

@mixin screen($media) {
  @if $media == 1330 {
    @media (max-width: 1330px) {@content};
  }
}

.weather {
    margin-top: 70px;
}

h1 {
    text-align: center;
    color: white !important;
    font-family: 'Open Sans', sans-serif;
    font-weight: 400;
}

.multiselect {
    width: 90%;
    margin: auto;
}

  .carousel {
    color: white;
```

```
}

.carousel-indicators li {
    outline: none;
}

.day {
    color: white;
    font-family: 'Open Sans', sans-serif;
    font-weight: 400;
    font-size: 18px;
    text-align: center;
    margin-top: 15px !important;
}

.VueCarousel-slide {
    text-align: center;
}

 .example-slide {
    align-items: center;
    // background-color: #666;
    color: #999;
    display: flex;
    font-size: 1.5rem;
    justify-content: center;
    min-height: 10rem;
    // flex-wrap: wrap;
    @include screen(1330) {
        flex-wrap: wrap;
    }
  }

.example-slide div {
    // width: 100%;
}

.VueCarousel-navigation-prev,
.VueCarousel-navigation-next {
    transition: all 0.2s;
}
.VueCarousel {
    &:hover {
        .VueCarousel-navigation-prev {
            transform: translateY(-50%) translateX(60%);
            transition: all 0.2s;
            &:focus {
                outline: none;
            }
        }
        .VueCarousel-navigation-next {
            transform: translateY(-50%) translateX(-75%);
            transition: all 0.2s;
            &:focus {
                outline: none;
            }
        }
    }
}

.VueCarousel-dot {
    &:focus {
        outline: none !important;
```

```
    }
}

.VueCarousel-navigation-button {
    top: 51.35% !important;
}

.day::first-letter {
    text-transform: uppercase;
}

.nav-carousel-next {
    width: 20px;
    height: 20px;
    border-radius: 20%;
    border-bottom: 5px solid white;
    border-right: 5px solid white;
    transform: rotate(-45deg);
}

.nav-carousel-prev {
    width: 20px;
    height: 20px;
    border-radius: 20%;
    border-bottom: 5px solid white;
    border-right: 5px solid white;
    transform: rotate(135deg);
}

.weather-temp {
    font-family: 'Open Sans', sans-serif ;
}

// ----------------- new carousel

.carousel {
    position: relative;
}

.carousel-inner {
    position: unset;
    bottom: 0;
    width: 100%;
    overflow: unset;
}

.carousel-caption {
    bottom: 0;
    top: 0;
}

.carousel-control-prev, .carousel-control-next {
    top: 200px;
}

.carousel-indicators {
    top: 345px;
    @include screen(1330) {
        top: 530px;
        margin-bottom: 90px;
    }
}
```

```
</style>
```

Icon.vue

```
<template>
  <div class="weather-icon-container">

    <div v-if="iconType === 'Atmosphere'" class="icon sun-shower">
        <div class="cloud"></div>
        <div class="sun">
            <div class="rays"></div>
        </div>
        <div class="rain"></div>
    </div>

    <div v-if="iconType === 'Thunderstorm'" class="icon thunder-storm">
        <div class="cloud"></div>
        <div class="lightning">
            <div class="bolt"></div>
            <div class="bolt"></div>
        </div>
    </div>

    <div v-if="iconType === 'Clouds'" class="icon cloudy">
        <div class="cloud"></div>
        <div class="cloud"></div>
    </div>

    <div v-if="iconType === 'Snow'" class="icon flurries">
        <div class="cloud"></div>
        <div class="snow">
            <div class="flake"></div>
            <div class="flake"></div>
        </div>
    </div>

    <div v-if="iconType === 'Clear'" class="icon sunny">
        <div class="sun">
            <div class="rays"></div>
        </div>
    </div>

    <div
    v-if="iconType === 'Drizzle' ||
    iconType === 'Rain'" class="icon rainy">
        <div class="cloud"></div>
        <div class="rain"></div>
    </div>

  </div>
</template>

<script>

export default {
  components: {},
  props: ['iconType']
  ,
  data () {
    return {}
  },
  mounted() {
```

```
      /* eslint-disable no-console */
      console.log(typeof(this.iconType),  111);

  }
}
</script>>

<style scoped lang="scss">

</style>
```

ProductApp.vue

```
<template>
  <div class="product-main">

    <div class="nav-bar"></div>
    <div class="product">
      <div class="product-image">
        <img :src="image">
      </div>

      <div class="product-info">
        <h1>{{ title }}</h1>
        <a :href="link" target="_blank">More products like this</a>
        <p v-if="inStock">In stock</p>
        <p v-else :class="{ outOfStock: !inStock }">Out of stock</p>
        <p>Shipping: {{ shipping }}</p>

        <ul>
          <li v-for="(detail, idx) in details" v-bind:key="idx">
            {{ detail }}
          </li>
        </ul>

        <div class="color-sock" v-for="(variant, index) in variants"
        v-bind:key="variant.variantId"
        :style="{ backgroundColor: variant.variantColor }"
        @mouseover="updateProduct(index)">
        </div>

        <button v-on:click="addToCart"
        :disabled="!inStock"
        :class="{disabledButton: !inStock}">Add to Cart</button>
        <button @click="removeFromCart">Remove from cart</button>

        <ProductTabs :reviews="reviews"/>

      </div>
    </div>

  </div>
</template>

<script>
import ProductTabs from '@/components/Task1/ProductTabs.vue'
import {eventBus} from '../../main.js'
export default {
  name: 'ProductApp',
  components: {
    ProductTabs
  },
```

```
  data() {
    return {
      premium: false,
      brand: "Vue Mastery",
      product: 'Socks',
      selectedVariant: 0,
      link: 'https://www.amazon.com/s/ref=nb_sb_noss?url=search-
alias%3Daps&field-keywords=socks',
      details: ['80% cotton', '20% polyester', 'Gender-neutral'],
      variants: [
        {
          variantId: 2234,
          variantColor: "green",
          variantImage: require('../../assets/green.jpg'),
          variantQuantity: 10
        },
        {
          variantId: 2235,
          variantColor: "blue",
          variantImage: require('../../assets/blue.jpg'),
          variantQuantity: 0
        }
      ],
      reviews: []
    }
  },
  methods: {
    addToCart() {
      this.$emit('add-to-cart', this.variants[this.selectedVariant].variantId)
    },
    removeFromCart: function() {
      this.$emit('remove-from-cart',
this.variants[this.selectedVariant].variantId)
    },
    updateProduct(index) {
      this.selectedVariant = index
    }
  },
  computed: {
    title() {
      return this.brand + ' ' + this.product
    },
    image() {
      return this.variants[this.selectedVariant].variantImage
    },
    inStock() {
      return this.variants[this.selectedVariant].variantQuantity
    },
    shipping() {
      if (this.premium) {
        return "Free"
      }
      return 2.99
    }
  },
  mounted() {
    eventBus.$on('review-submitted', productReview => {
      this.reviews.push(productReview)
    })
  }
 }
</script>
```

```
<!-- Add "scoped" attribute to limit CSS to this component only -->
<style scoped lang="scss">

  .product {
    display: flex;
    flex-flow: wrap;
    padding: 1rem;
  }

  img {
    border: 1px solid #d8d8d8;
    width: 70%;
    margin: 15px 40px 40px 40px;
    box-shadow: 0px .5px 1px #d8d8d8;
  }

  .product-image {
    width: 80%;
  }

  .product-image,
  .product-info {
    margin-top: 10px;
    width: 45%;
  }

  .color-box {
    width: 40px;
    height: 40px;
    margin-top: 5px;
  }


  .color-sock {
    cursor: pointer;
    width: 40px;
    height: 40px;
    margin-top: 5px;
  }

  button {
    margin-top: 30px;
    border: none;
    background-color: #1E95EA;
    color: white;
    height: 40px;
    min-width: 100px;
    margin: 30px 10px 20px 0px;
    font-size: 14px;
  }

  .disabledButton {
    background-color: #d8d8d8;
  }

  .outOfStock {
    text-decoration: line-through;
  }
</style>
```

ProductReview.vue

```
<template>
    <div>
        <form class="review-form" @submit.prevent="onSubmit">

            <p v-if="errors.length">
                <b>Please correct the following error(s):</b>
                <ul>
                    <li v-for="(error, idx) in errors" :key="idx">{{ error
}}</li>
                </ul>
            </p>

            <p>
                <label for="name">Name:</label>
                <input id="name" v-model="name">
            </p>

            <p>
                <label for="review">Review:</label>
                <textarea id="review" v-model="review"></textarea>
            </p>

            <p>
                <label for="rating">Rating:</label>
                <select id="rating" v-model.number="rating">
                    <option>5</option>
                    <option>4</option>
                    <option>3</option>
                    <option>2</option>
                    <option>1</option>
                </select>
            </p>

            <p>
                <input type="submit" value="submit">
            </p>

        </form>
        <!-- <input v-model="name"> -->
    </div>
</template>

<script>
import {eventBus} from '../../main.js'
export default {
  name: 'ProductReview',
  data () {
    return {
        name: null,
        review: null,
        rating: null,
        errors: []
    }
  },
  methods: {
      onSubmit() {
          if(this.name && this.review && this.rating) {
            let productReview = {
                name: this.name,
                review: this.review,
                rating: this.rating
            }
            eventBus.$emit('review-submitted', productReview)
```

```
            this.name = null
            this.review = null
            this.rating = null
          }
            else {
                if(!this.name) this.errors.push("Name required.")
                if(!this.review) this.errors.push("Review required.")
                if(!this.rating) this.errors.push("Rating required.")
            }
      }
  }
}
</script>

<style scoped lang="scss">

  .review-form {
    width: 400px;
    padding: 20px;
    margin: 40px;
    border: 1px solid #d8d8d8;
  }

  input {
    width: 100%;
    height: 25px;
    margin-bottom: 20px;
  }

  textarea {
    width: 100%;
    height: 60px;
  }

</style>
```

ProductTabs.vue

```
<template>
    <div>
        <span class="tab"
        :class="{ activeTab: selectedTab === tab}"
        v-for="(tab, index) in tabs" :key="index"
        @click="selectedTab = tab">
        {{ tab }}</span>

        <div v-show="selectedTab === 'Reviews'">
          <h2 class="h2-review">Reviews</h2>
          <p v-if="!reviews.length">There are no reviews yet.</p>
          <ul>
            <li v-for="(review, idx) in reviews" :key="idx">
              <p>Name: {{ review.name }}</p>
              <p>Comment: {{ review.review }}</p>
              <p>Rating: {{ review.rating }}</p>
            </li>
          </ul>
        </div>

        <ProductReview
        v-show="selectedTab === 'Make a Review'"/>

    </div>
```

```
</template>

<script>
import ProductReview from '@/components/Task1/ProductReview.vue'
export default {
  props: {
      reviews: {
          type: Array,
          required: true
      }
  },
  components: {
      ProductReview
  },
  name: 'ProductTabs',
  data () {
      return {
          tabs: ['Reviews', 'Make a Review'],
          selectedTab: 'Reviews'
      }
  }
}
</script>

<style scoped lang="scss">

  .tab {
    margin-right: 14px;
    cursor: pointer;
    background: #1e95ea;
    padding: 10px;
    color: white;
  }

  .activeTab {
    color: black;
    font-weight: bold;
    text-decoration: underline;
  }

  .h2-review {
    margin-top: 25px;
  }

</style>
```

User.vue

```
<template>
    <div class="ui card">
        <div class="image">
          <img :src="user.avatar_url">
        </div>
        <div class="content">
          <a :href="`https://github.com/${username}`"
class="header">{{user.name}}</a>
          <div class="meta">
            <span class="date">Joined in {{user.created_at}}</span>
          </div>
          <div class="description">
            {{user.bio}}
          </div>
```

```
        </div>
        <div class="extra content">
          <a :href="`https://github.com/${username}?tab=followers`">
            <i class="user icon"></i>
            {{user.followers}} Friends
          </a>
        </div>
    </div>
</template>

<script>
/* eslint-disable no-console */
export default {
  name: 'api-ex',
  props: {
        username: {
          type: String,
          required: true
        }
      },
      data() {
        return {
          user: {}
        }
      },
      created() {
        this.$axios.get(`https://api.github.com/users/${this.username}`)
          .then(response => {
            console.log(response.data);

            this.user = response.data
          })
      }
}
</script>

<!-- Add "scoped" attribute to limit CSS to this component only -->
<style lang="scss">
</style>
```

config.js

```
export default {
    apiKeyGeo: '935d1398-e768-4a75-b157-89945e7e66b3',
    geoApi: 'https://geocode-maps.yandex.ru/1.x/',
    apiKeyWeather: '6b15515ee8eefb096f7b1d9bd692dfee',
    weatherApi: 'https://api.openweathermap.org/data/2.5/forecast?',
    weatherApiOneDay: 'https://api.openweathermap.org/data/2.5/weather?q='
}
```

Package.json

```
{
  "name": "task2",
  "version": "0.1.0",
  "private": true,
  "scripts": {
    "serve": "vue-cli-service serve",
    "build": "vue-cli-service build",
    "lint": "vue-cli-service lint"
  },
  "dependencies": {
```

```
    "axios": "^0.19.0",
    "bootstrap": "^4.4.1",
    "bootstrap-vue": "^2.1.0",
    "core-js": "^3.4.3",
    "es6-promise": "^4.2.8",
    "moment": "^2.24.0",
    "semantic-ui-card": "^2.3.1",
    "vue": "^2.6.10",
    "vue-carousel": "^0.18.0",
    "vue-moment": "^4.1.0",
    "vue-multiselect": "^2.1.6",
    "vue-router": "^3.1.3",
    "vuex": "^3.1.2"
  },
  "devDependencies": {
    "@vue/cli-plugin-babel": "^4.1.0",
    "@vue/cli-plugin-eslint": "^4.1.0",
    "@vue/cli-plugin-router": "^4.1.0",
    "@vue/cli-service": "^4.1.0",
    "babel-eslint": "^10.0.3",
    "eslint": "^5.16.0",
    "eslint-plugin-vue": "^5.0.0",
    "node-sass": "^4.12.0",
    "sass-loader": "^8.0.0",
    "vue-template-compiler": "^2.6.10"
  },
  "eslintConfig": {
    "root": true,
    "env": {
      "node": true
    },
    "extends": [
      "plugin:vue/essential",
      "eslint:recommended"
    ],
    "rules": {},
    "parserOptions": {
      "parser": "babel-eslint"
    }
  },
  "browserslist": [
    "> 1%",
    "last 2 versions"
  ]
}
```

main.js

```
import Vue from 'vue'

import axios from 'axios'
Vue.prototype.$axios = axios
import VueCarousel from 'vue-carousel';
Vue.use(VueCarousel);

import BootstrapVue from 'bootstrap-vue'
Vue.use(BootstrapVue);

import moment from 'moment'
Vue.prototype.moment = moment
moment.locale('ru');
```

```
import App from './App.vue'
import router from './router'

import store from './store'

Vue.config.productionTip = false

export const eventBus = new Vue()

new Vue({
  router,
  store,
  render: h => h(App)
}).$mount('#app')
```

App.vue

```
<template>
  <div id="app">
    <keep-alive> <!-- Кэширует компоненты -->
    <router-view/>
    </keep-alive>
  </div>
</template>

<style lang="scss">
@import "./styles/styles.css";

#app {
  width: 100vw;
  height: 100vh;
  font-family: tahoma;
  color:#282828;
  margin: 0px;
}

body {
  margin: 0px;
}
</style>
```

Router.js

```
import Vue from 'vue'
import VueRouter from 'vue-router'
import Home from '../views/MainTask.vue'
import VuexExample from '@/views/AppVuex.vue'
import Chapter1 from '@/views/Multiselect.vue'
import Chapter2 from '@/views/Navigation.vue'
import Chapter3 from '@/views/Pagination.vue'
import Navigation from '@/views/WeatherIcon.vue'  // ---> тогда в пути вместо
import вставляю 'Navigation'

Vue.use(VueRouter)

const routes = [
{
    path: '/',
    name: 'WeatherServise',
    component: Home
},
{
    path: '/vuex',
```

```
    name: 'VuexExample',
    component: VuexExample
},
{
    path: '/Multiselect,
    name: Multiselect,
    component: Multiselect
},
{
    path: '/Pagination,
    name: Pagination,
    component: Pagination
},
{
    path: '/WeatherIcon,
    name: WeatherIcon,
    component: WeatherIcon
},
{
    path: '/Navigation',
    name: 'Navigation',
    component: Navigation
},
{
    path: '/weather',
    name: 'weather',
    component: Home
}]

const router = new VueRouter({
  mode: 'history',
  base: process.env.BASE_URL,
  routes
})

export default router
```

store.js

```
import Vue from 'vue'
import Vuex from 'vuex'
import cart from './modules/cart'
import products from './modules/products'
import createLogger from '../../../src/plugins/logger'

Vue.use(Vuex)

const debug = process.env.NODE_ENV !== 'production'

export default new Vuex.Store({
  modules: {
    cart,
    products
  },
  strict: debug,
  plugins: debug ? [createLogger()] : []
})
```

weatherStore.js

```
import shop from '../../api/weather'
```

```
const state = {
  items: [],
  checkoutStatus: null
}

// getters
const getters = {
  searchStates: (state, getters, rootState) => {
    return state.items.map(({ id, quantity }) => {
      const product = rootState.products.all.find(product => state.id === id)
      return {
        title: search.title,
        coordinate: search.coordinate
      }
    })
  },

  totalAim: (state, getters) => {
    return getters.totalAim.reduce((total, search) => {
      return total + search.name * search.adress
    }, 0)
  }
}

// actions
const actions = {
  checkout ({ commit, state }, search) {
    const savedAdress = [...state.items]
    commit('setCheckoutStatus', null)
    commit('setLon', { items: [] })
    weather.api(
      search,
      () => commit('setCheckoutStatus', 'successful'),
      () => {
        commit('setCheckoutStatus', 'failed')
        // rollback to the api saved before sending the request
        commit('setLonAndLat', { items: savedSearch })
      }
    )
  },

  sendCoordinate ({ state, commit }, coordinate) {
    commit('setCheckoutStatus', null)
    if (item.inventory > 0) {
      const container = state.items.find(item => item.id === search.id)
      if (!coordinate) {
        commit('pushCoordinate ',
        { id: coordinate.id })
      } else {
        commit('reset', item)
      }
      // remove 1 item from stock
      commit('products/'pushCoordinate, { id: item.id }, { root: true })
    }
  }
}

// mutations
const mutations = {
  pushCoordinate (state, { id }) {
    state.items.push({
      id,
    })
```

```
  },

  resetCoordinate (state, { id }) {
    const coordinate = state.items.find(item => item.id === id)
    coordinate.item = []
  },

  setCartCoord (state, { items }) {
    state.items = items
  },

  setCheckoutStatus (state, status) {
    state.checkoutStatus = status
  },

  resetAll (state, status) {
    state = ...state
  }
}

export default {
  namespaced:
  true,
  state,
  getters,
  actions,
  mutations
}
```