Міністерство освіти і науки України

Харківський національний університет імені В.Н. Каразіна

Факультет комп’ютерних наук

Кафедра моделювання систем і технологій

**Пояснювальна записка**

до дипломної роботи

бакалавра

на тему: «Реалізація веб-застосунку для візуалізації

багатовимірних даних з застосуванням алгоритму t-SNE»

Виконав: студент 4 курсу, групи ЗКС-41

напряму підготовки (спеціальності)

6.050101 – комп’ютерні науки

Мишенін Антон Сергійович

Керівник ст. викладач Мішин О.В.

Рецензент ст. викладач Діденко Є.В.

Харків - 2018

РЕФЕРАТ

Дипломна робота містить 93 сторінки, 3 таблиці, 11 рисунків, список використаної літератури з 12 найменувань, 1 додаток.

Мета цієї дипломної роботи полягає в створенні веб-застосунку для візуалізації багатовимірних даних за допомогою алгоритму зниження їх розмірності t-distributed stochastic neighbor embedding (t-SNE).

Для написання цієї дипломної роботи, було використано Amazone Web Services, а саме S3 Bucket, Lambda і API Gateway для створення бекенду, і React.js, Redux, Webpack – для фронтенду. Методами дослідження виступають: формалізація даних, аналіз та синтез даних.

Ми ознайомилися з алгоритмами та методами візуалізації багатовимірних даних, їх перевагами, недоліками та особливостями. Оскільки найбільшу зацікавленість в нас викликав алгоритм t-SNE (t-distributed stochastic neighbor embedding), тому було вирішено використовувати його за основу нашого застосунку. Незважаючи на його математичну складність, як самого алгоритму, так і його імплементації, нам вдалося не тільки працювати з ним, використовуючи готовий node.js модуль в режимі «чорної скриньки», [[1]](#footnote-1)але й ознайомитись з ним і максимально спрощено, але в той самий час точно, описати принцип дії алгоритму t-SNE.

Було вирішено спроектувати програмний продукт у формі веб-застосунку. Такий підхід надає доступ до застосунку всім користувачам мережі інтернет, незважаючи на операційну систему та характеристики їх обчислювального пристрою. Що безперечно забезпечує кросплатформеність застосунку і потенційно збільшує його аудиторію, в порівнянні з застосунками, виконаними з використанням підходу програмного забезпечення, що потребує встановлення на комп’ютері чи мобільному пристрої.

Напрямки розвитку веб-застосунку включають, але не обмежуються наступним:

* додання автоматичних тестів та механізмів continuous integration
* впровадження системи замовлення результатів
* поліпшення механізмів аналізу даних
* покращення інтерфейсу користувача

Детальніше про перспективи розвитку читати розділ 6.

Ключові слова: візуалізація, візуалізація даних, візуалізація багатовимірних даних, багатовимірні дані, алгоритм t-SNE, t-distributed stochastic neighbor embedding, машинне навчання, зниження розмірності, зниження розмірності багатовимірних даних.

ABSTRACT

Thesis contains 98 pages, 3 tables, 11 figures, list of used literature from 12 titles, 1 appendix.

The purpose of this thesis is to create a web application for visualizing multidimensional data using a t-distributed stochastic neighbor embedding (t-SNE) algorithm.

Bucket, Lambda and API Gateway for backend, and React.js, Redux, Webpack for the front-end were used. Research methods are: data formalization, analysis and data synthesis.

We got acquainted with algorithms and methods of visualization of multidimensional data, their advantages, disadvantages and features. Since the t-SNE algorithm (t-distributed stochastic neighbor embedding) caused us the greatest interest, it was decided to use it as the basis of our application. Despite its mathematical complexity, both the algorithm itself and its implementation, we managed not only to work with it, using the finished node.js module in the "black box" [[2]](#footnote-2)mode, but also get acquainted with it and as simple as possible, but at the same time for sure, describe the principle of the t-SNE algorithm.

It was decided to design a software product in the form of a web application. This approach provides access to the application to all users of the Internet, regardless of the operating system and characteristics of their computing device. Which unquestionably provides cross-platform application and potentially increases its audience, compared with applications implemented using a software approach that requires installation on a computer or mobile device.

The directions for developing a web application include, but are not limited to, the following:

* adding of automatic tests and mechanisms of continuous integration
* implementation of the system of ordering the results
* improved data analysis mechanisms
* Improvement of the user interface

Read more about the development prospects of section 6.

Key words: visualization, data visualization, multidimensional data visualization, multidimensional data, t-SNE algorithm, t-distributed stochastic neighbor embedding, machine learning, dimension reduction, multidimensional data dimension reduction.

ЗМІСТ

[ВСТУП 7](#_Toc514256841)

[РОЗДІЛ 1. АНАЛІЗ СУЧАСНИХ МЕТОДІВ ВІЗУАЛІЗАЦІЇ ДАНИХ 11](#_Toc514256842)

[РОЗДІЛ 2. ПРИНЦИП ДІЇ АЛГОРИТМУ T-SNE 14](#_Toc514256843)

[РОЗДІЛ 3. ДОСЛІДЖУВАНА МНОЖИНА ДАНИХ 16](#_Toc514256844)

[РОЗДІЛ 4. АРХІТЕКТУРА І РЕАЛІЗАЦІЯ ЗАСТОСУНКУ 17](#_Toc514256845)

[4.1. АРХІТЕКТУРА І РЕАЛІЗАЦІЯ БЕКЕНДУ 18](#_Toc514256846)

[4.1.1. ПРИНЦИП ДІЇ AWS LAMBDA ФУНКЦІЇ MAIN LAMBDA 20](#_Toc514256847)

[4.1.2. ПРИНЦИП ДІЇ AWS LAMBDA ФУНКЦІЇ WEATHER LAMBDA 22](#_Toc514256848)

[4.1.3. ПРИНЦИП ДІЇ AWS LAMBDA ФУНКЦІЇ t-SNE LAMBDA 23](#_Toc514256849)

[4.2. АРХІТЕКТУРА І РЕАЛІЗАЦІЯ ФРОНТЕНДУ 24](#_Toc514256850)

[4.2.1. КОНФІГУРАЦІЯ ІНСТУМЕНТУ ЗБІРКИ ПРОЕКТУ WEBPACK 25](#_Toc514256851)

[4.2.2. КОМПОНЕНТ APP 27](#_Toc514256852)

[4.2.3. КОМПОНЕНТ LOCATIONMAP 30](#_Toc514256853)

[4.2.4. КОМПОНЕНТ PERIODSPICKER 31](#_Toc514256854)

[4.2.5. КОМПОНЕНТ STARTBUTTON 34](#_Toc514256855)

[4.2.6. КОМПОНЕНТ TSNECHART. 35](#_Toc514256856)

[4.2.7. КОМПОНЕНТ OUTPUT. 36](#_Toc514256857)

[4.3. МОБІЛЬНА ВЕРСІЯ ЗАСТОСУНКУ. 37](#_Toc514256858)

[РОЗДІЛ 5. ІНСТРУКЦІЯ КОРИСТУВАЧА 39](#_Toc514256859)

[РОЗДІЛ 6. ПЕРСПЕКТИВИ РОЗВИТКУ 42](#_Toc514256860)

[6.1. АВТОМАТИЧНІ ТЕСТИ, CONTINUOUS INTEGRATION 42](#_Toc514256861)

[6.2. ВПРОВАДЖЕННЯ СИСТЕМИ ЗАМОВЛЕННЯ РЕЗУЛЬТАТІВ. 42](#_Toc514256862)

[6.3. ПОКРАЩЕННЯ АНАЛІЗУ ПОГОДНИХ ДАНИХ 43](#_Toc514256863)

[6.4. ПОЧАТИ ПРЯМУ СПІВПРАЦЮ З КОМАНДОЮ WORLD WEATHER ONLINE 44](#_Toc514256864)

[6.5. ПОКРАЩЕННЯ ФРОНТЕНДУ 44](#_Toc514256865)

[ВИСНОВКИ 45](#_Toc514256866)

[СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 49](#_Toc514256867)

[ДОДАТОК А. 51](#_Toc514256868)

# ВСТУП

Одним з найефективніших методів аналізу даних є їх візуалізація. Полягає вона на представленні даних у вигляді зображення, що зосереджено на властивостях всієї множини даних, а не окремих її екземплярах, що забезпечує можливість відстежувати тенденції, робити прогнози та взагалі збільшує продуктивність під час дослідження даних.

Приведемо достатньо простий приклад аналізу даних за допомогою візуально методу. Розглянемо таблицю 1, вона показує зміни цін на акції компанії Motorola Solutions Inc. на проміжку часу з 9 квітня до 4 травня 2018 року. Попри те, що в цій таблиці містяться всі необхідні для аналізу дані, їх дослідження дуже проблематичне і скоріше за все в такій формі представлення може бути здійснено тільки комп’ютерними методами. Для дослідження людиною, дані в такій формі є майже непридатними.

Таблиця 1 – Таблиця цін на акції компанії Motorola Solutions Inc. у відповідні дні.

|  |  |
| --- | --- |
| Date | Open |
| May 04, 2018 | 107.96 |
| May 03, 2018 | 107.52 |
| May 02, 2018 | 108.70 |
| May 01, 2018 | 109.83 |
| Apr 30, 2018 | 110.40 |
| Apr 27, 2018 | 109.85 |
| Apr 26, 2018 | 109.59 |
| Apr 25, 2018 | 109.25 |
| Apr 24, 2018 | 109.97 |
| Apr 23, 2018 | 110.85 |
| Apr 17, 2018 | 109.25 |
| Apr 16, 2018 | 107.95 |
| Apr 13, 2018 | 108.38 |
| Apr 12, 2018 | 108.24 |
| Apr 11, 2018 | 106.95 |
| Apr 10, 2018 | 106.01 |
| Apr 09, 2018 | 104.71 |

Порівняємо таку форму представлення даних з тією, що зображено на рисунку 1, а саме графіком, на осі абсцис якого відкладені дати, а на осі ординат ціни за акції. В цій формі ми не бачимо конкретних значень ціни в певний день, однак, ми отримаємо можливість робити висновки на основі цієї множини даних. Наприклад, ми бачимо, що в період з 6 квітня по 18 квітня відбувалося значне зростання в ціні на акції з незначним спадом в періоді близько 13 квітня. З 18 квітня до 30 квітня відбувався незначний спад, що з 30 квітня перейшло в падіння до 2 травня, і було продовжено незначним зростанням. На майбутнє можна зробити висновок, що якщо тенденції не зміняться, то ціна буде коливатися в районі 107 доларів США за акцію з відхиленням в більшу та меншу сторону до 5 доларів США.

Безперечно, ми могли б прийти до таких висновків, використовуючи табличне представлення, але це б значно ускладнило процес, зменшило продуктивність і збільшило час дослідження.

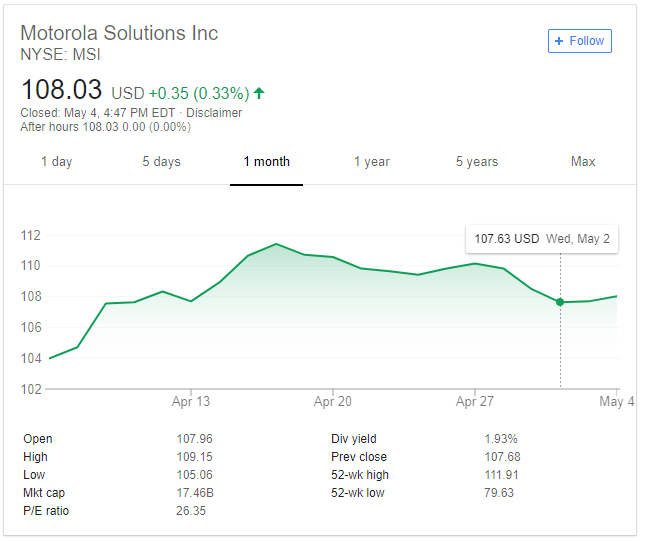


Рисунок 1 – Графік зміни цін на акції компанії Motorola Solutions Inc.

На жаль, не завжди так легко представити дані в графічному вигляді так, як в нашому попередньому прикладі. Розглянемо таблицю 2, вона показує прогноз погоди на 9 травня 2018 погодинно для Берліна, штат Меріленд, США. За параметри досліджуваного нами явища, а саме погоди, виступають наступні величини час, температура, вологість та швидкість вітру, всього 4. І тут постає питання, як зобразити такі дані, адже кожен запис в таблиці являє собою точку чотирьох вимірного простору, який ми не те що зобразити не можемо, ми не можемо його навіть уявити і навіть зрозуміти, оскільки ніколи з таким не стикалися в житті.

Таблиця 2 – Прогноз погоди на 9 травня 2018 для міста Берлін, штат Меріленд, США

|  |  |  |  |
| --- | --- | --- | --- |
| Час | Температура (ºC) | Вологість  (%) | Швидкість вітру (км/г) |
| 00:00 | 12 | 10 | 8 |
| 01:00 | 12 | 10 | 8 |
| 02:00 | 12 | 10 | 8 |
| 03:00 | 11 | 10 | 9 |
| 04:00 | 11 | 10 | 10 |
| 05:00 | 11 | 10 | 11 |
| 06:00 | 13 | 10 | 11 |
| 07:00 | 14 | 10 | 11 |
| 08:00 | 16 | 10 | 11 |
| 09:00 | 17 | 10 | 12 |
| 10:00 | 19 | 10 | 13 |
| 11:00 | 20 | 10 | 13 |
| 12:00 | 21 | 10 | 13 |
| 13:00 | 21 | 10 | 13 |
| 14:00 | 21 | 10 | 13 |
| 15:00 | 21 | 10 | 12 |
| 16:00 | 21 | 10 | 12 |
| 17:00 | 21 | 10 | 11 |
| 18:00 | 19 | 10 | 10 |
| 19:00 | 17 | 10 | 9 |
| 20:00 | 14 | 10 | 7 |
| 21:00 | 14 | 10 | 7 |
| 22:00 | 13 | 0 | 7 |
| 23:00 | 12 | 0 | 7 |

Але від цього, дослідження даних в табличній формі не стало простіше, тому нам потрібно знайти спосіб, як візуалізувати багатовимірні дані на площину.

Отже, зважаючи на наявну проблему, метою дипломної роботи ми визначили створення веб-застосунку, що дозволить користувачу візуалізувати введені ним багатовимірні дані. Для досягнення цієї мети, за завдання проекту, ми визначили наступне:

1. Ознайомитись з методиками візуалізації багатовимірних даних, зокрема алгоритмом T-SNE.
2. Визначити досліджувану множину даних.
3. Розробити архітектуру застосунку.
4. Обрати технології, за допомогою яких буде реалізовано застосунок.
5. Реалізувати застосунок.

# РОЗДІЛ 1. АНАЛІЗ СУЧАСНИХ МЕТОДІВ ВІЗУАЛІЗАЦІЇ ДАНИХ

Оскільки візуалізація даних може відбуватися на площині у постаті двовимірного рисунка, або в просторі у постаті тривимірної моделі, маючи справу з даними вищих розмірностей, постає питання їх зниження. В нашому другому прикладі зі вступу, кожен запис являє собою точку чотиривимірного простору. Таким чином, завдання візуалізації багатовимірних даних зводиться до зниження їх розмірності, бо зображення даних, що мають розмірність 1, 2 або навіть 3 є тривіальною роботою.

Існує багато методів і алгоритмів для зниження розмірності множини багатовимірних даних, але усі вони потрапляють в одну з двох категорій: відбір ознак (англ. feature selection) і виділення ознак (англ. feature extraction).

Методи категорії відбір ознак полягають на тому, що ми відбираємо ті величини, які є суттєвими для нас і ігноруємо ті, що такими не є. Приведемо приклад, якщо ми візьмемо таблицю 2 і видалимо з неї стовпці «Вологість» та «Швидкість вітру», вважаючи ці дані для нас не істотними, то отримаємо таблицю 1.1, яку вже можна представити у вигляді звичайного графіку (рисунок 1.1).

Таблиця 1.1 – Прогноз погоди на 9 травня 2018 для міста Берлін, штат Меріленд, США, що ігнорує не суттєві для нас ознаки.

|  |  |
| --- | --- |
| Час | Температура (ºC) |
| 00:00 | 12 |
| 01:00 | 12 |
| 02:00 | 12 |
| 03:00 | 11 |
| 04:00 | 11 |
| 05:00 | 11 |
| 06:00 | 13 |
| 07:00 | 14 |
| 08:00 | 16 |
| 09:00 | 17 |
| 10:00 | 19 |
| 11:00 | 20 |
| 12:00 | 21 |
| 13:00 | 21 |
| 14:00 | 21 |
| 15:00 | 21 |
| 16:00 | 21 |
| 17:00 | 21 |
| 18:00 | 19 |
| 19:00 | 17 |
| 20:00 | 14 |
| 21:00 | 14 |
| 22:00 | 13 |
| 23:00 | 12 |

Рисунок 1.1 – графік, що відображає інформацію про погоду на 9 травня 2018 для міста Берлін, штат Меріленд, США,

Звісно, цей приклад – це велике узагальнення. Багато алгоритмів попадає під цю категорію, і кожен з них по-своєму обирає які ознаки вважати важливими, а які ігнорувати. Якісь опираються на предметну область, в якій працює дослідник, інші вибираю ті ознаки, де найбільше різноманіття значень.

Другий клас алгоритмів – виділення однак, полягає на тому, що в результаті не містяться початкові величини, а лише результивні, які були вирахувані на основі початкових. При такому підході не ігнорується жодна з величин і кожна з них має свій вплив на кінцевий результат. Розглянемо найпростіший приклад роботи такого алгоритму. Припустимо, ми маємо графік, що зображений на рисунку 1.2, і наше завдання полягає в тому, щоб знизити розмірність до одновимірного простору, тобто осі. Для цього проведемо пряму, що паралельна вектору найбільших змін величини. Спроектуємо точки на цю пряму. Ця пряма і є нашим одновимірним простором. Кожній точці на прямій буде відповідати число, що не є ані початковою координатою x, ані початковою координатою y, однак, вони, координати, безпосередньо і в рівній мірі визначили значення нової ознаки.

Рисунок 1.2 – Приклад переведення двовимірних даних в одновимірні за допомогою підходу виділення ознак.

Важко виділити який підхід є кращім, на сам перед, тому що обидва підходи знайшли своє застосування при вирішенні тих чи інших завдань. Однак, зважаючи на те, що реалізація другого підходу є більш цікавою с точки зору програмного продукту, ми обираємо цей підхід. Слід згадати, що виділення однак, це цілий клас алгоритмів, до якого належать наступні алгоритми:

* PCA – Principal Component Analysis
* t-SNE – t-Distributed Stochastic Neighbor Embedding
* Isomap
* Local multidimensional scaling

і багато інших алгоритмів, кожен з яких є ефективний для певного типу даних.

Зважаючи на те, що ми були дуже зацікавлені алгоритмом t-SNE, ми його обрали за основу нашого застосунку.

# РОЗДІЛ 2. ПРИНЦИП ДІЇ АЛГОРИТМУ T-SNE

При узагальненому розгляді принципу дії алгоритму t-SNE, його роботу можна розділити на 2 етапи:

1. Побудова розподілу ймовірностей для кожної пари багатовимірних об’єктів, таким чином, що схожі об’єкти мають високу ймовірність бути згрупованими, а несхожі – малу.
2. Побудова подібного розподілу ймовірностей на відповідній низько вимірній мапі, а між самими розподілами мінімізується відстань Кульбака — Лейблера[[3]](#footnote-3), враховуючи місцеперебування точок.

Оскільки з теоретичної точки зору, і перший, і другий етап цього алгоритму описані в працях Laurens van der Maaten і Geoffrey Hinton настільки чітко і повно, наскільки це можливо, і через надзвичайно складний і довгий їх опис, ознайомимось з алгоритмом на практиці.

Уявимо собі, що ми маємо двовимірний простір, тобто площину, на якій зображені точки (рисунок 2.1). Кожна точка являє собою сукупність двох величин, так само як в прикладі з цінами на акції Motorola Solution Inc. Наша задача – перевести двовимірні точки в одновимірні, тобто точку на прямій.

Переходимо до першого етапу, тобто побудови розподілу ймовірностей для кожної пари багатовимірних об’єктів, таким чином, що схожі об’єкти мають високу ймовірність бути згрупованими, а несхожі – малу.

1. Беремо одну з точок і визначаємо її як точку зацікавленості (рисунок 2.1). Визначимо відстані від неї до інших точок.
2. Підставимо відстань як аргумент функції нормального розподілу Гауса, отримані значення назвати немасштабованими коефіцієнтами схожості.
3. Масштабувати ці коефіцієнти, поділивши їх на суму всіх немасштабованих коефіцієнтів схожості.
4. Повторяти ці дії поки кожна з точок не побуває у ролі точки зацікавлення.

Рисунок 2.1 – Випадкова сукупність двовимірних даних і відзначена точка зацікавленості

1. Сформувати таблицю в стовпцях і рядках якої будуть точки, а в комірках – масштабовані коефіцієнти схожості між двома точками, при цьому масштабований коефіцієнт схожості точки на саму себе вважати рівним 0.

На цьому завершується перший етап. Другий етап, як і було зазначено вище, – це побудова подібного розподілу ймовірностей на відповідній низько вимірній мапі, а між самими розподілами мінімізується відстань Кульбака — Лейблера , враховуючи місцеперебування точок.

1. В низько вимірному, в нашому випадку на осі, випадково розставляємо точки у кількості, що відповідає кількості точок у початковому просторі.
2. Так само, як і на першому етапі формуємо матрицю масштабованих коефіцієнтів схожості, однак на цей раз використовуємо не нормальний розподіл Гауса, а t-розподіл Стюарда, звідси і назва t-distributed.
3. Далі, ми переміщаємо точки на осі, таким чином, щоб перша і друга матриці були приблизно тотожні.

Звичайно, це дещо спрощена інтерпретація алгоритму, але вона максимально розкриває суть того, що відбувається.

# РОЗДІЛ 3. ДОСЛІДЖУВАНА МНОЖИНА ДАНИХ

Оскільки ми розроблюємо застосунок для візуалізації даних, було б добре визначити які саме дані ми хочемо візуалізувати, адже дані характеризує не тільки їх структура, але й предметна область до якої вони відносяться, а вже характеристики аналізованих даних вже визначають вимоги до конкретного програмного продукту.

За досліджувану множини даних було вирішено використовувати історичні погодні дані за задані користувачем періоди і в заданих користувачем місцях. В останні роки, проблеми зміни клімату стають все більш і більш актуальними, з нарощуванням виробництва у країнах, що розвиваються – Китай, Індія, Нігерія, Бангладеш, та інших та введенням нових екологічних стандартів в розвинутих країнах Європи та Північної Америки, в тому числі підписання міжнародної Паризької умови і виходу з неї Сполучених Штатів Америки. Багато, що відбувається в галузі протидії змінам клімату, тому необхідність відстежувати ці зміни є нагальною. До того ж, метеорологічні дані є дуже складними і є одним з класичних прикладів багатовимірних даних, тому для таких даних візуалізація можлива тільки після зменшення їх розмірності.

На жаль, ми не володіємо метеорологічною базою даних, що б могла постійно оновлюватись і зберігати дані про погодні умови багато років тому. Однак, нам пощастило, бо навіть якщо ми не маємо такої бази даних, це не означає, що її не має ніхто. Багато метеорологічних організацій дають доступ до своїх баз даних та прогностичних можливостей через API. В нашій роботі, ми користувались API, що надала компанія World Weather Online у режимі пробного користування з обмеженим функціоналом і періодом користування, за що ми дуже їм вдячні.

# РОЗДІЛ 4. АРХІТЕКТУРА І РЕАЛІЗАЦІЯ ЗАСТОСУНКУ

Розглянувши принцип роботи алгоритму, що був взятий за основу нашого застосунку, і дані, що ним будуть оброблюватись, ми можемо перейти до розгляду самого програмного продукту, а саме – до його архітектури (рисунок4.1). Перш за все, наш застосунок є веб-застосунком, це означає, що він складається з двох частин клієнтської частини, тобто фронтенду, і серверної частини, тобто бекенду.

Фронтенд – це програма, що виконується в браузері користувача і відповідає за побудову інтерфейсу користувача, збір даних від користувача, відправку до сервера запиту, що містить введені користувачем дані.

Бекенд – це програма чи комплекс програм, що виконуються на віддаленому сервері і відповідає за прийняття запиту, здійснення обчислень, в нашому випадку зниження розмірності даних, відправлення відповіді з результатом.

Комунікація між цими двома частинами відбувається за протоколом HTTP.

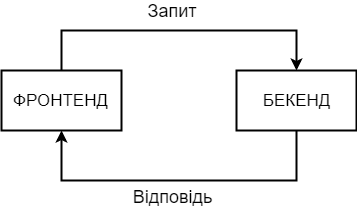


Рисунок 4.1 – Діаграма взаємодії клієнтської частини (фронтенда) з серверною (бекенда).

## 4.1. АРХІТЕКТУРА І РЕАЛІЗАЦІЯ БЕКЕНДУ

Вході вивчення способів написання серверної частини програмного забезпечення, ми зупинились на новітньому підході, запропонованому компанією Amazon у 2014 році, а саме AWS Lambda. Цей підхід передбачає, що розробник пише свій код у вигляді функції, що потім виконує вже запущений, відповідним чином налаштований і підтримуваний сервер компанії Amazon. Таким чином, з розробника знімається відповідальність за налаштування програмного сервера, і він може цілком зосередитись на написанні самого функціоналу.

До переваг використання AWS Lambda функцій можна віднести:

1. відсутність необхідності налаштовувати програмний сервер, підтримувати його працездатність
2. автоматична масштабованість залежно від кількості запитів до того чи іншого функціоналу
3. менша кількість фінансових затрат в порівнянні з програмним сервером, оскільки AWS Lambda функція виконується тільки тоді, коли до неї надійшов запит, а програмний сервер виконується завжди, тому плата у випадку AWS Lambda відбувається за кількість запитів (до 1000000 запитів на місяць безкоштовно), а в випадку програмного сервера за час, який він працює, всупереч кількість запитів

Як і кожен підхід, AWS Lambda має і свої недоліки:

1. відсутність контролю за середовищем виконання функції
2. більш складна архітектура застосунків
3. не можливість зберігання стану застосунку

Але зважаючи на те, що ці недоліки не є критичними для нашого завдання, цей підхід себе виправдовує.

В роботі серверної частини застосунку можна виділити наступні етапи:

1. прийняття запитання
2. отримання даних про погоду в певному місці в певний період з API, що надала компанія World Weather Online
3. підготовка даних для t-SNE алгоритму
4. зниження розмірності за допомогою t-SNE алгоритму
5. видача результату

Зважаючи на це, архітектура бекенду передбачає існування трьох AWS Lambda функцій (рисунок 4.2):

1. Weather Lambda – на основі отриманих даних посилає запит до World Weather Online API запит, отримає результат, приводить до виду, що може бути оброблений алгоритмом t-SNE
2. t-SNE Lambda – знижує розмірність отриманих даних і повертає результат
3. Main Lambda – отримує запит від фронтенду, пов’язує Weather Lambda і t-SNE Lambda, передає дані для обробки Weather Lambda і отримує у відповідь дані придатні до обробки алгоритмом t-SNE, потім передає дані до t-SNE Lambda і отримує у відповідь дані, що вона посилає у відповідь на запит від інтерфейсу користувача

Схему взаємодії трьох функцій можна описати наступною діаграмою (рисунок4.1.2). Як можна побачити з діаграми, до бекенду надходить запит, що приймає функція Main Lambda. Вона викликає функцію Weather Lambda з заздалегідь підготовленими даними з запиту. Функція Weather Lambda оброблює ці дані і повертає як результат дані про погоду в формі придатній для алгоритму t-SNE. Далі функція Main Lambda викликає функцію t-SNE Lambda, передаючи їй дані, що були отримані з Weather Lambda. Функція t-SNE повертає оброблені за допомогою алгоритму t-SNE дані, що потім Main Lambda повертає на фронтенд як відповідь.

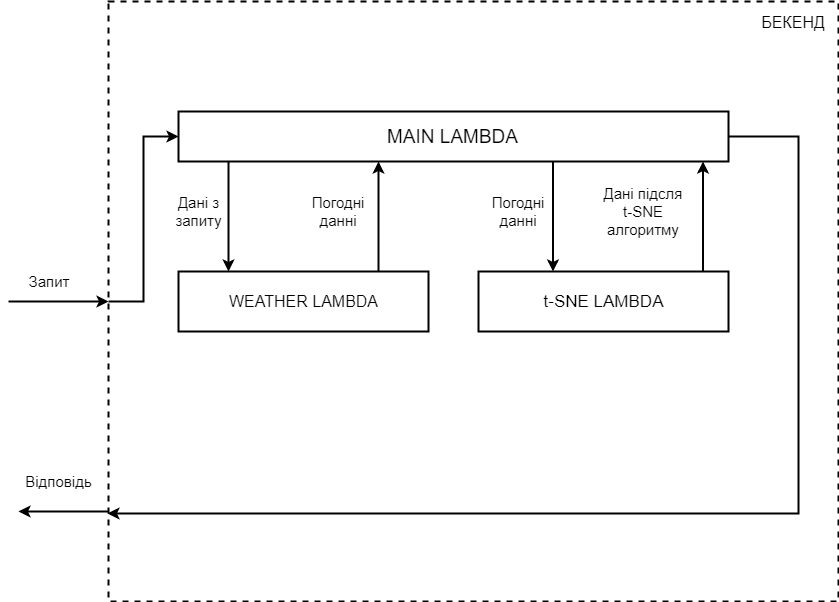


Рисунок 4.2 – Діаграма, що відображає архітектуру бекенду

### 4.1.1. ПРИНЦИП ДІЇ AWS LAMBDA ФУНКЦІЇ MAIN LAMBDA

Функція Main Lambda – є головним елементом архітектури бекенду, що пов’язує між собою всі інші функції, керує потоком даних між ними і їх викликом, і в кінцевому результаті повертає відповідь до фронтенду.

Як можна побачити з початкового коду (рисунок 4.3), функція Main Lambda приймає запит від фронтенду, в цьому запиті містяться координати того місця, для якого ми визначаємо погодні дані, та масив періодів, за які ми ці дані беремо. Для кожного періоду, ми асинхронно викликаємо Weather Lambda з даними (місце і період). Коли вона повертає дані стосовно всіх періодів, ми об’єднуємо всі ці дані в один масив і надсилаємо його до t-SNE Lambda, яка повертає нам результат зниження розмірності даних. Цей результат ми надсилаємо як відповідь з кодом HTTP 200, що означає успіх проведеної операції.

|  |
| --- |
| **const** AWS = require('aws-sdk');  AWS.config.update({region: 'eu-central-1'});    **const** lambda = **new** AWS.Lambda();  module.exports.main = ({body}, context, callback) => {  body = JSON.parse(body);    Promise  .all(  body.periods  .map(period => **new** Promise((resolve, reject) => {  **const** params = {  FunctionName: 'weather-dev-weather',  InvocationType: 'RequestResponse',  Payload: JSON.stringify({  lat: body.location.lat,  lng: body.location.lng,  start: period.start,  end: period.end  })  };    lambda.invoke(params, (error, data) => {  **if** (error) {  reject(error);  } **else** {  **const** {Payload} = data;  resolve(JSON.parse(Payload));  }  });    }))  )  .then(weatherData => **new** Promise((resolve, reject) => {  **const** params = {  FunctionName: 'tsne-dev-tsne',  InvocationType: 'RequestResponse',  Payload: JSON.stringify(weatherData)  };    lambda.invoke(params, (error, data) => {  **if** (error) {  reject(error);  } **else** {  **const** {Payload} = data;  resolve(JSON.parse(Payload));  }  });  }))  .then(result => {  **const** response = {  statusCode: 200,  headers: {  'Access-Control-Allow-Origin': '\*'  },  body: JSON.stringify(result),  };    callback(**null**, response);  })  }; |

Рисунок 4.3 – Початковий код Main Lambda

### 4.1.2. ПРИНЦИП ДІЇ AWS LAMBDA ФУНКЦІЇ WEATHER LAMBDA

Призначення функції Weather Lambda полягає в тому, щоб повертати погодні дані у відповідній для алгоритму t-SNE формі. Вона здійснює запит до наданого компанією World Weather Online API з даними, що були отримані з функції Main Lambda. Після отримання даних, вони трансформовані в необхідну для алгоритму t-SNE форму (рисунок 4.4).

Слід зазначити, що оскільки API надане в безкоштовній версії існують певні обмеження:

* дозволяється не більше 500 запитів до API на день
* перший запит в дні приходить з помилкою, через внутрішні проблеми API
* кожен період вважається окремим запитом

|  |
| --- |
| **const** request = require('request-promise');    module.exports.main = (event, context, callback) => {  **const** {lat, lng, start, end} = event;  request(`https:*//api.worldweatheronline.com/premium/v1/past-weather.ashx?key=${process.env.WEATHER\_API\_KEY}&format=json&q=${lat},${lng}&date=${start}&enddate=${end}`)*  .then(\_response => {  **const** data = JSON.parse(\_response).data  .weather  .map(day => day.hourly  .map(hour => [  parseFloat(hour.tempC),  parseFloat(hour.windspeedKmph),  parseFloat(hour.winddirDegree),  parseFloat(hour.humidity),  parseFloat(hour.visibility),  parseFloat(hour.pressure),  parseFloat(hour.cloudcover),  ])  )  .reduce((accumulator, current) => {  **return** [...accumulator, ...current];  }, []);    callback(**null**, data);  })  .**catch**((error) => {  callback(error);  });  }; |

Рисунок 4.4 – Початковий код функції Weather Lambda

### 4.1.3. ПРИНЦИП ДІЇ AWS LAMBDA ФУНКЦІЇ t-SNE LAMBDA

Це, мабуть, найголовніша функція, тому що вона виконує зниження розмірності даних, що були отримані в ході роботи функції Weather Lambda. Для цього ми використали node.js модуль під назвою tsne-js авторства користувача сервісу github.com scienceai. Ми просто передаємо отримані з Main Lambda функції дані в відповідний метод node.js модуля і отримаємо масив зниженої розмірності для кожного періоду, після чого ми повертаємо масив цих масивів назад (рисунок 4.5).

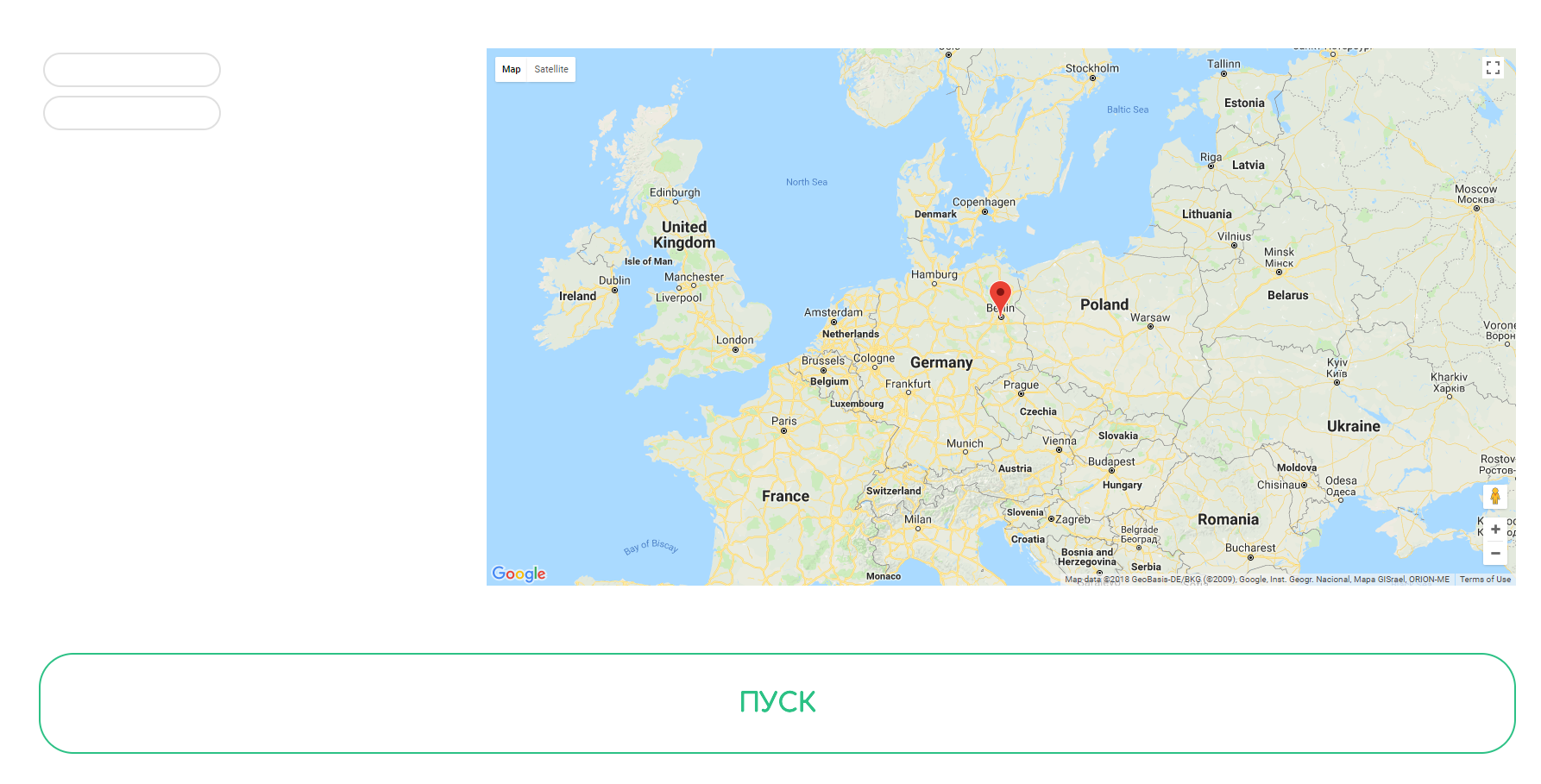
|  |
| --- |
| **const** TSNE = require(‘tsne-js’);  **const** model = **new** TSNE({  dim: 2,  perplexity: 30.0,  earlyExaggeration: 4.0,  learningRate: 100.0,  nIter: 1000,  metric: ‘euclidean’  });    module.exports.main = (event, context, callback) => {  callback(**null**, event.map(period => {  model.init({  data: period,  type: ‘dense’  });  **for**(let k = 0; k < 500; k++) {  tsne.step();  }  **return** tsne.getSolution();  }));  }; |

Рис 4.5 – Початковий код функції t-SNE Lambda

## 4.2. АРХІТЕКТУРА І РЕАЛІЗАЦІЯ ФРОНТЕНДУ

Не менш важливою складовою кожного застосунку є клієнтська частина, тобто фронтенд. Саме фронтенд взаємодіє з користувачем, надсилає запит до бекенду і отримує відповідь від нього, а отже відповідальний за її правильне зображення перед очима користувача.

В нашому програмному продукті, візуально, інтерфейс вводу даних складається з трьох частин: гугл-мапи – для вибору місця, поля вводу дат – для вводу періодів, кнопки для запуску обчислень (рисунок 4.6).



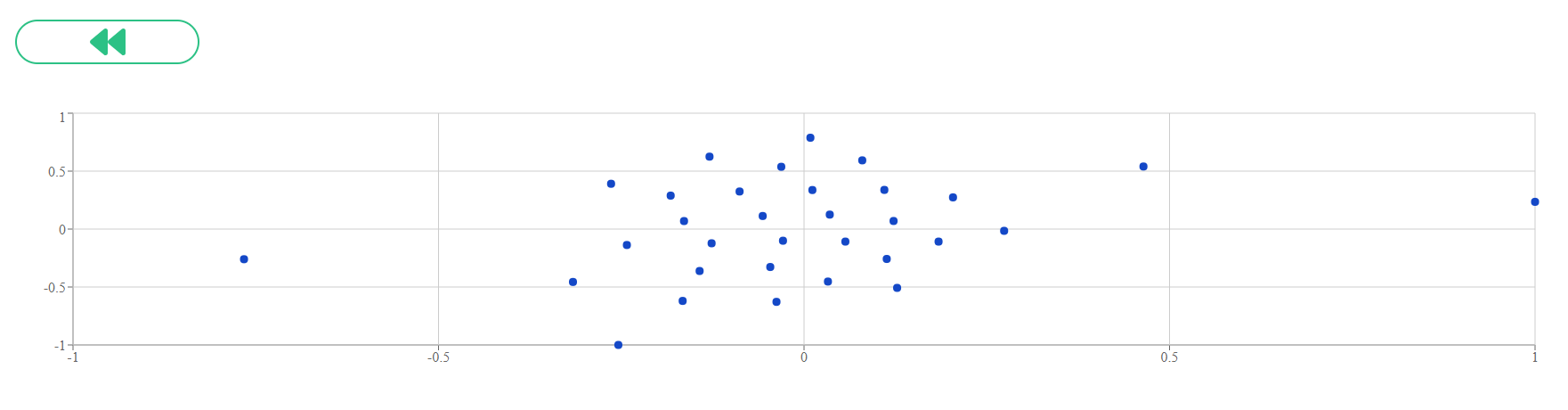
Кнопка запуску обчислень

Поля вводу дат

Гугл-мапа для вибору місця

Рисунок 4.6 – Інтерфейс користувача при вводі даних

Результати обчислень виводять в інтерфейсі виводу даних, що виглядає наступним чином (рисунок 4.7).

Рисунок 4.7 – Інтерфейс користувача при виводі даних

Слід зазначити, що для кожного періоду будується свій t-SNE графік. Такий підхід допомагає порівнювати різні періоди і визначати наскільки вони відрізняються одне від одного.

Для імплементації фронтенду ми вирішили використовувати бібліотеку React.js і Redux, оскільки з цими бібліотеками ми ще не були знайомі, а отже робота з ними стала викликом і відмінним поштовхом до саморозвитку.

З метою хостінгу клієнтської частини застосунку, було використано AWS S3 Bucket – онлайн служба, що призначена для зберігання файлів, але в додаток до цього, вона здатна здійснювати хостінг статичного контенту, яким і є зібраний фроненд проект. Такий підхід також виключає необхідність у програмному сервері для видачі статичних сторінок.

### 4.2.1. КОНФІГУРАЦІЯ ІНСТУМЕНТУ ЗБІРКИ ПРОЕКТУ WEBPACK

Всі Reactjs-застосунки є досить складними проектами, що складаються з багатьох компонентів, в додаток до цього, при використанні бібліотеки Redux в проекті з’являються функції, що керують так званим «станом» – об’єктом, що зберігає дані в застосунку. Також, зважаючи на те, що React-компоненти пишуться на ECMAScript 6, в початковому стані цей код не є розпізнаваним браузером. Тому, перед тим, як завантажувати його в інтернет (не має значення на сервер чи AWS S3 Bucket чи тощо), його необхідно трансформувати в сукупність наступних файлів:

* bundle.js – сукупність всього ECMAScript 6 коду переведеного до стандарту ECMAScript 5 за допомогою Babel і зв’язаного в один файл, також служить для надання веб-сторінці стилів
* index.html – файл, що є документом, в якому ECMAScript 5 код здійснює свою роботу
* /assets/ – зображення та інші медіа-файли, що були використані в проекті

Інструмент збірки Webpack потребує відповідної конфігурації в корені проекту в файлі webpack.conf.js (рисунок 4.8). Від неї залежить, яким чином буде працювати webpack.

|  |
| --- |
| **const** path = require('path');  **const** CopyWebpackPlugin = require('copy-webpack-plugin');    **const** DIST\_DIR = path.resolve(\_\_dirname, 'dist');  **const** SRC\_DIR = path.resolve(\_\_dirname, 'src');    **const** config = {  entry: `${SRC\_DIR}/app/index.jsx`,  output: {  path: DIST\_DIR,  filename: 'bundle.js',  publicPath: '/',  },  module: {  loaders: [  {  test: */\.jsx?/*,  include: SRC\_DIR,  loader: 'babel-loader',  query: {  presets: ['react', 'es2015', 'stage-2'],  },  },  {  test: [/\.scss$/, /\.css$/],  loader: 'style-loader',  },  {  test: [/\.scss$/, /\.css$/],  loader: 'css-loader',  },  {  test: */\.scss$/*,  loader: 'sass-loader',  },  ],  },  plugins: [  **new** CopyWebpackPlugin([  {  from: './src/index.html',  to: './index.html',  },  {  from: './assets/img/\*',  to: '.'  }  ])  ],  resolve: {  extensions: ['.js', '.jsx'],  },  };    module.exports = config; |

Рисунок 4.8 – Конфігурація webpack

### 4.2.2. КОМПОНЕНТ APP

Батьківським компонентом для всього інтерфейсу є компонент App, саме в ньому або безпосередньо, або опосередковано знаходяться всі інші компоненти. Також він відіграє велику роль при контролюванні об’єкту-стану застосунку, оскільки він є компонентом, що має безпосередній доступ до нього. Всі інші компоненти мають доступ до стану тільки через компонент App. Такі компоненти, як App в термінології бібліотеки Redux ще називають контейнерами (рисунок 4.9).

|  |
| --- |
| import React from 'react';  import {connect} from 'react-redux';  import LocationMap from "../components/LocationMap";  import chooseLocation from "../redux/reducers/location/actions/chooseLocation";  import MobileTabs from "../components/MobileTabs";  import changeTab from "../redux/reducers/tabs/actions/changeTab";  import PeriodsPicker from "../components/PeriodsPicker";  import setStartDate from "../redux/reducers/periods/actions/setStartDate";  import setEndDate from "../redux/reducers/periods/actions/setEndDate";  import StartButton from "../components/StartButton";  import getData from "../redux/reducers/weather/actions/getData";  import Output from "../components/Output";  import inputOutputSwitch from "../redux/reducers/input-output-switcher/actions/inputOutpuSwitch";    **function** arraysEqual(a, b) {  **if** (a === b) **return** **true**;  **if** (a === **null** || b === **null**) **return** **false**;  **if** (a.length !== b.length) **return** **false**;    **for** (let i = 0; i < a.length; ++i) {  **if** (a[i] !== b[i]) **return** **false**;  }  **return** **true**;  }    class App extends React.Component {  constructor(props) {  super(props);  }    componentDidMount() {  **this**.props.inputOutputSwitch({  width: window.innerWidth,  mode: 'INPUT'  });    **if** (parseInt(window.innerWidth) < 768) {  **this**.props.changeTab([**true**, **false**, **false**]);  } **else** {  **this**.props.changeTab([**true**, **true**, **true**]);  }    window.addEventListener("resize", () => {  **this**.props.inputOutputSwitch({  width: window.innerWidth,  mode: **this**.props.inputOutputSwitcher.mode  });    **if** ((parseInt(window.innerWidth) < 768)) {  **if** (arraysEqual(**this**.props.tabs, [**true**, **true**, **true**])) {  **this**.props.changeTab([**true**, **false**, **false**]);  }  } **else** {  **this**.props.changeTab([**true**, **true**, **true**]);  }  });  }    render() {  **return** (  <div className="container">  {**this**.props.tabs[0] && **this**.props.inputOutputSwitcher.\_input && <LocationMap  chooseLocation={**this**.props.chooseLocation}  location={**this**.props.location}  />}  {**this**.props.tabs[1] && **this**.props.inputOutputSwitcher.\_input && <PeriodsPicker  setStartDate={**this**.props.setStartDate}  setEndDate={**this**.props.setEndDate}  periods={**this**.props.periods}  />}  {**this**.props.tabs[2] && **this**.props.inputOutputSwitcher.\_output && <Output  inputOutputSwitch={**this**.props.inputOutputSwitch}  inputOutputSwitcher={**this**.props.inputOutputSwitcher}  weather={**this**.props.weather}  />}  {**this**.props.inputOutputSwitcher.\_input && <StartButton  getData={**this**.props.getData}  inputOutputSwitch={**this**.props.inputOutputSwitch}  inputOutputSwitcher={**this**.props.inputOutputSwitcher}  location={**this**.props.location}  periods={**this**.props.periods}  />}  <MobileTabs  changeTab={**this**.props.changeTab}  tabs={**this**.props.tabs}    getData={**this**.props.getData}  location={**this**.props.location}  periods={**this**.props.periods}    inputOutputSwitcher={**this**.props.inputOutputSwitcher}  inputOutputSwitch={**this**.props.inputOutputSwitch}  />  </div>  );  }  }    **const** mapStateToProps = state => {  **return** {  location: state.location,  tabs: state.tabs,  periods: state.periods,  weather: state.weather,  inputOutputSwitcher: state.inputOutputSwitcher  };  };    **const** mapDispatchToProps = dispatch => {  **return** {  chooseLocation: (payload) => {  dispatch(chooseLocation(payload));  },  changeTab: (payload) => {  dispatch(changeTab(payload));  },  setStartDate: (payload) => {  dispatch(setStartDate(payload));  },  setEndDate: (payload) => {  dispatch(setEndDate(payload));  },  getData: (payload) => {  dispatch(getData(payload));  },  inputOutputSwitch: (payload) => {  dispatch(inputOutputSwitch(payload));  },  }  };      export **default** connect(mapStateToProps, mapDispatchToProps)(App); |

Рисунок 4.9 – Початковий код компонента App.

### 4.2.3. КОМПОНЕНТ LOCATIONMAP

Компонент LocationMap призначений для того, щоб користувач за допомогою нього міг обрати будь-яке місце на мапі, а в об’єкті-стану були збережені його координати. Надалі ці координати будуть відправлені на бекенд.

Для зображення гугл-мапи, була використана бібліотека react-google-maps авторства Tom Chen (рисунок 4.10). Цей компонент асоційований з частиною стану location, а також з однойменним редюсером [[4]](#footnote-4)(рисунок 4.11, рисунок 4.12).

|  |
| --- |
| import React from 'react';  import {compose, withProps} from "recompose"  import {GoogleMap, Marker, withGoogleMap, withScriptjs} from "react-google-maps"    **const** chooseLocation = (action) => {  **return** **function** (params) {  action({  lat: params.latLng.lat(),  lng: params.latLng.lng(),  });  }  };    **const** LocationMap = compose(  withProps({  googleMapURL: "https://maps.googleapis.com/maps/api/js?key=AIzaSyA4QDUJ4RRuq4ajSWIWzCUlRTPI4guQzGM&v=3.exp&libraries=geometry,drawing,places",  loadingElement: <div className="map"/>,  containerElement: <div className="map"/>,  mapElement: <div style={{height: `100%`}}/>,  }),  withScriptjs,  withGoogleMap  )((props) => <GoogleMap  defaultZoom={5}  defaultCenter={props.location}  onClick={chooseLocation(props.chooseLocation)}  >  <Marker position={props.location}/>  </GoogleMap>  );    export **default** LocationMap; |

Рисунок 4.10 – Початковий код компонента LocationMap

|  |
| --- |
| **const** location = (state = {lat: 52.52, lng: 13.405}, action) => {  **switch** (action.type) {  **case** 'CHOOSE\_LOCATION':  state = action.payload;  **break**;  }  **return** state;  };    export **default** location; |

Рисунок 4.11 – Початковий код редюсера location

|  |
| --- |
| **const** chooseLocation = (payload) => {  **return** {  type: 'CHOOSE\_LOCATION',  payload  };  };    export **default** chooseLocation; |

Рисунок 4.12 – Початковий код дії [[5]](#footnote-5)chooseLocation для редюсера location

### 4.2.4. КОМПОНЕНТ PERIODSPICKER

Вибір періодів, якими ми зацікавлені, здійснюють за допомогою компонента PeriodsPicker. Він складається зі змінної кількості полів початкової та кінцевої дати. За потреби, ця кількість може збільшуватись.

Цей компонент написаний з застосуванням бібліотеки react-datepicker авторства користувача сервісу github.com HackerOne (рисунок 4.13). Компонент асоційований з частиною стану periods, а також однойменним редюсером і двома діями до нього (рисунок 4.14 – 4.16).

|  |
| --- |
| import React from ‘react’;    import DatePicker from ‘react-datepicker’;  import ‘react-datepicker/dist/react-datepicker.css’;  import moment from ‘moment’;  moment.locale(‘uk’);    **const** onChangeStart = (key, action) => {  **return** (date) => {  action({key, date});  };  };    **const** onChangeEnd = (key, action) => {  **return** (date) => {  action({key, date});  };  };  **const** PeriodsPicker = (props) => {  **return** (  <div className=»periods-picker»>  {  props.periods.map((period, index) => (  <div key={3 \* index}>  <DatePicker  selected={period.start}  selectsStart  startDate={period.start}  endDate={period.end}  onChange={onChangeStart(index, props.setStartDate)}  key={3 \* index + 1}  />    <DatePicker  selected={period.end}  selectsEnd  startDate={period.start}  endDate={period.end}  onChange={onChangeEnd(index, props.setEndDate)}  key={3 \* index + 2}  />  </div>  ))  }  </div>  );  };    export **default** PeriodsPicker; |

Рисунок 4.13 – Початковий код компонента PeriodsPicker

|  |
| --- |
| **const** periods = (state = [{start: **null**, end: **null**}], action) => {  **switch** (action.type) {  **case** 'SET\_START\_DATE': {  let newState = [...state];  newState[action.payload.key].start = action.payload.date;  state = newState;  }  **break**;    **case** 'SET\_END\_DATE': {  **if** (action.payload.key === state.length - 1) {  let newState = [...state];  newState[action.payload.key].end = action.payload.date;  newState.push({  start: **null**,  end: **null**  });  state = newState;  } **else** {  let newState = [...state];  newState[action.payload.key].end = action.payload.date;  state = newState;  }  }  **break**;  }  **return** state;  };    export **default** periods; |

Рисунок 4.14 – Початковий код редюсера periods

|  |
| --- |
| **const** setStartDate = (payload) => {  **return** {  type: 'SET\_START\_DATE',  payload  };  };    export **default** setStartDate; |

Рисунок 4.15 – Початковий код дії setStartDate для редюсера periods

|  |
| --- |
| **const** setEndDate = (payload) => {  **return** {  type: 'SET\_END\_DATE',  payload  };  };  export **default** setEndDate; |

Рисунок 4.16 – Початковий код дії setEndDate для редюсера periods

### 4.2.5. КОМПОНЕНТ STARTBUTTON

Щоб почати обчислення, користувач натискає відповідну кнопку, яка виконана у вигляді компонента StartButton (рисунок 4.17). І вже цей компонент за посередництвом відповідного редюсера відправляє запит, і після цього змінює сторінку вводу даних на сторінку їх виводу, щоб можна було побачити результат.

|  |
| --- |
| import React from 'react';    **const** StartButton = (props) => {  **return** (  <div className="start-button">  <button onClick={() => {  props.getData({  location: props.location,  periods: props.periods  .filter(period => period.start && period.end)  .map(period => ({start: period.start.format('YYYY-MM-DD'), end: period.end.format('YYYY-MM-DD')}))  });  props.inputOutputSwitch({  width: props.inputOutputSwitcher.width,  mode: 'OUTPUT'  });  }}>ПУСК  </button>  </div>  );  };    export **default** StartButton; |

Рисунок 4.17 – Початковий код компонента StartButton

Редюсер приймає дані з відповіді на запит і трансформує їх в форму придатну для компонента, що виводить графік на сторінці виводу даних.

|  |
| --- |
| **const** weather = (state = [], action) => {  **switch**(action.type) {  **case** 'GET\_DATA\_FULFILLED': {  state = action.payload.data.map(period => period.map(point => ({x: point[0], y: point[1]})));  } **break**;  }  **return** state;  };    export **default** weather; |

Рисунок 4.18 – Початковий код редюсера weather

Відповідальність за здійснення запиту до бекенду покладено на дію getData, що повертає в редюсер відповідь у вигляді об’єкту типу Promise[[6]](#footnote-6).

|  |
| --- |
| **const** axios = require('axios');    **const** getData = (payload) => {  **return** {  type: 'GET\_DATA',  payload: axios.post('https://vfurc2tlyg.execute-api.eu-central-1.amazonaws.com/dev/result', payload)  };  };    export **default** getData; |

Рисунок 4.19 – Початковий код дії getData для редюсера weather

### 4.2.6. КОМПОНЕНТ TSNECHART.

Дані по кожному періоду зображені на окремому t-SNE графіку, за який відповідає компонент TSNEChart. Він отримує дані з батьківського компонента і підставляє їх в компоненти, що відповідають за зображення графіку. Ці компоненти були взяті з бібліотеки recharts авторства компанії Recharts Group. При цьому компонент TSNEChart керує вирівнюванням елементів всередині нього за допомогою стилів SASS і динамічних стилів, згенерованих в ході виконання коду.

|  |
| --- |
| import React from 'react';  import {ResponsiveContainer, ScatterChart, XAxis, YAxis, CartesianGrid, Scatter} from 'recharts';    **function** getRandomColor() {  let letters = '0123456789ABCDEF';  let color = '#';  **for** (let i = 0; i < 6; i++) {  color += letters[Math.floor(Math.random() \* 16)];  }  **return** color;  }      **const** TSNEChart = (props) => {  **const** style = {  gridRow: `${4 + 2 \* props.index}/${5 + 2 \* props.index}`  };  **return** (  <div className="chart" style={style}>  <ResponsiveContainer>  <ScatterChart>  <XAxis dataKey={'x'} type="number" name='x'/>  <YAxis dataKey={'y'} type="number" name='y'/>  <CartesianGrid />  <Scatter name='TSNE data' data={props.data} fill={getRandomColor()}/>  </ScatterChart>  </ResponsiveContainer>  </div>  );  };    export **default** TSNEChart; |

Рисунок 4.20 – Початковий код компонента TSNEChart

### 4.2.7. КОМПОНЕНТ OUTPUT.

Оскільки компонент TSNEChart зображає лише один графік, а нам необхідно вивести одразу декілька, для всіх періодів, а отже необхідно для кожного періоду вивести свій компонент TSNEChart. Цей факт призводить до необхідності обернути графіки в батьківський елемент, в нашому випадку – компонент Output.

|  |
| --- |
| import React from 'react';    import FontAwesomeIcon from '@fortawesome/react-fontawesome';  import {faBackward} from '@fortawesome/fontawesome-free-solid';    import TSNEChart from "./TSNEChart";    **const** Output = (props) => {  **const** style = {  gridTemplateRows: `50px 50px 50px ${props.weather.map(() => '300px 50px').join(' ')}`  };  **return**(  <div className="output" style={style}>  <div className="back">  <button onClick={props.inputOutputSwitch.bind(**null**, {  width: props.inputOutputSwitcher.width,  mode: 'INPUT'  })}>  <FontAwesomeIcon icon={faBackward} size="lg"/>  </button>  </div>  {  props.weather.map((weatherForOnePeriod, index) => {  **return** (  <TSNEChart  key={index}  index={index}  data={weatherForOnePeriod}  />  );  })  }  </div>  );  };    export **default** Output; |

Рисунок 4.21 – Початковий код компонента Output.

## 4.3. МОБІЛЬНА ВЕРСІЯ ЗАСТОСУНКУ.

В сучасному світі все більше і більше користувачів відкривають програми не за допомогою персональних комп’ютерів, а за допомогою переносних пристроїв: мобільних телефонів, планшетів, тощо. Однією з особливостей таких пристроїв є зменшена ширина екрану, тому програмний продукт в сучасному світі повинен бути пристосований до мобільних пристроїв.

Наш програмний продукт має наступний вигляд на мобільних пристроях (рисунок 4.22).



Рисунок 4.22 – мобільний вигляд застосунку.

На горі розміщена панель закладок, за допомогою якої можна вводити різні частини даних (місце, періоди), а також перейти на сторінку результатів.

Це було здійснено за допомогою технології CSS Grid та додаткових компонентів (рисунок 4.23).

|  |
| --- |
| import React from 'react';    import FontAwesomeIcon from '@fortawesome/react-fontawesome';  import {faMapMarkerAlt, faPlay} from '@fortawesome/fontawesome-free-solid';  import {faClock} from '@fortawesome/fontawesome-free-regular';    **const** MobileTabs = (props) => {  **return** (  <div className="mobile-tabs">  <div className={`tab ${props.tabs[0] && 'current'}`}  onClick={props.changeTab.bind(**null**, [**true**, **false**, **false**])}>  <FontAwesomeIcon icon={faMapMarkerAlt} size="lg"/>  <div>Місце</div>  </div>  <div className={`tab ${props.tabs[1] && 'current'}`}  onClick={props.changeTab.bind(**null**, [**false**, **true**, **false**])}>  <FontAwesomeIcon icon={faClock} size="lg"/>  <div>Періоди</div>  </div>  <div className={`tab ${props.tabs[2] && 'current'}`} onClick={() => {  props.changeTab([**false**, **false**, **true**]);  props.getData({  location: props.location,  periods: props.periods  .filter(period => period.start && period.end)  .map(period => ({start: period.start.format('YYYY-MM-DD'), end: period.end.format('YYYY-MM-DD')}))  });  props.inputOutputSwitch({  width: props.inputOutputSwitcher.width,  mode: 'OUTPUT'  });  }}>  <FontAwesomeIcon icon={faPlay} size="lg"/>  <div>Результат</div>  </div>  </div>  )  };    export **default** MobileTabs; |

Рисунок 4.23 – Початковий код компонента MobileTabs.

# РОЗДІЛ 5. ІНСТРУКЦІЯ КОРИСТУВАЧА

Попри те, що інтерфейс користувача доволі інтуїтивно зрозумілий, все одно в цьому розділі було вирішено описати послідовність дій користувача.

1. Перейти на сторінку застосунку <http://climate-analyzer.s3-website.eu-central-1.amazonaws.com/>
2. Вибрати місце на мапі, натиснув на нього (рисунок 5.1).



Ми клікнули на випадкове місто в Чехії

Рисунок 5.1 – мапа, на якій обрано випадкове місто в Чехії

1. Вибрати періоди, за які треба отримати дані. Для цього треба звернути свою увагу на пару полів вводу дат. Верхнє поле відповідає початку періоду, а нижнє – його кінцю. При натисканні відкривається календар для вибору дати. Коли період вибраний, тобто задані, і початкова, і кінцева дата, з’являється поля для ще одного періоду (рисунок 5.2).

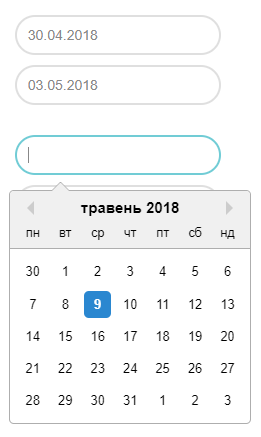
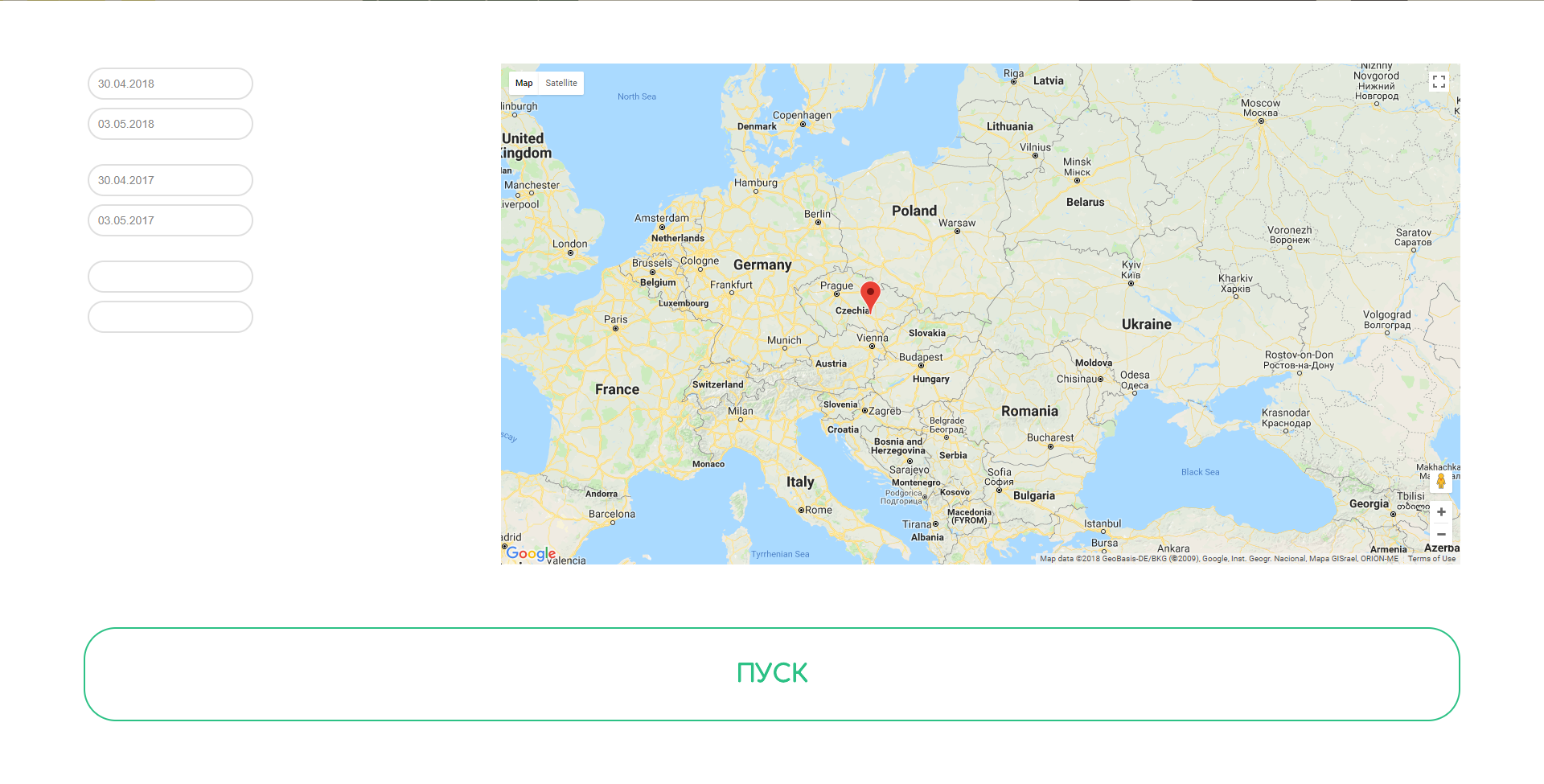


Рисунок 5.2 – поля для вибору періодів.

1. Натиснути кнопку пуск (рисунок 5.3).

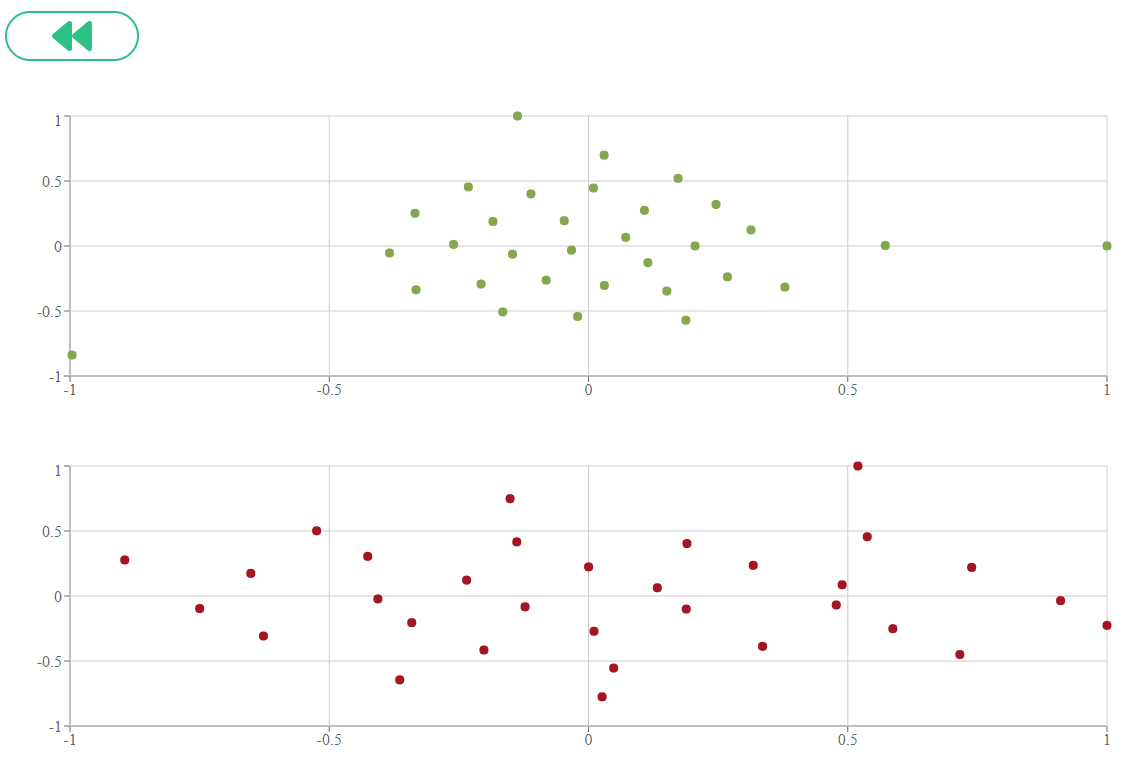


Натисни на мене

Рисунок 5.3 – інтерфейс з виділеною кнопкою запуску.

1. Перед собою користувач бачить сторінку результатів, на якій зображені графіки для кожного періоду і кнопку назад до сторінки вводу даних

(рисунок 5.4).



Кнопка назад

Графік для періоду

30.04.18 – 03.05.18

Графік для періоду

30.04.17 – 03.05.17

Рисунок 5.4 – сторінка виводу результатів.

# РОЗДІЛ 6. ПЕРСПЕКТИВИ РОЗВИТКУ

Як і кожний програмний продукт, наш застосунок потребує не стояти на місці і далі розвиватися, покращувати свій інтерфейс користувача, свої обчислювальні можливості, а також додавати новий функціонал. Саме тому, в цьому розділі ми розкажемо о тому, які обновлення ми маємо в планах.

## 6.1. АВТОМАТИЧНІ ТЕСТИ, CONTINUOUS INTEGRATION

Важливою складовою сучасних програмних продуктів є автоматичні тести, написані для них. Тести допомагають розробникам програмного забезпечення бути впевненим в тому, що при зміні чи доданню одного функціоналу, весь інший функціонал є працездатним, при цьому не змушуючи у ручному режимі перевіряти кожну функцію.

Для тестування бекенду буде використано фреймворк для тестів Cucumber.js, оскільки він дає змогу записувати не тільки самі тести, але і use cases для певних функцій, що спрощує їх документацію. Тести для фронтенду будуть написані за допомогою технології Protractor і Selenium.

Ці тести будуть запускатися за допомогою налаштованої системи continuous integration, яка після отриманні позитивного результату тестів дозволить злиття гілки функціоналу з основною гілкою розробки на репозиторії (github.com), збудує проект і завантажить до інтернету, фронтенд – до AWS S3 Bucket, функції – до сервісу AWS Lambda. Якщо результати тестів будуть негативними, то злиття гілок забороняється, а попередній стан проекту в інтернеті відновлюється. Таким чином, користувач завжди буде мати робочу версію проектую

## 6.2. ВПРОВАДЖЕННЯ СИСТЕМИ ЗАМОВЛЕННЯ РЕЗУЛЬТАТІВ.

В ході роботи над застосунком ми зіткнулися з наступним обмеженням, пов’язаним зі складністю обчислень, яку має алгоритм t-SNE, кожен заданий період повинен бути менше ніж14 днів. Пов’язано це з тим, що для кожного дня World Weather Online API повертає 8 значень погодних вимірювань протягом всього дня. Це означає, на обробку алгоритмом t-SNE поступає масив розміром 112 рядків і 8 стовпців (кількість показників для кожного періоду). При збільшені кількості днів збільшується кількість елементів масиву, а відповідно і час обробки. І попри те, що сама AWS Lambda може виконувати обчислення до 5 хвилин, сервіс що надає адресу в інтернеті API Gateway дає максимум 30 секунд на виконання запиту. Тому, загалом обмеження за часом 30 секунд. До того, хочеться зазначити, що і користувачу буде неприємно очікувати такий довгий час.

Тому ми пропонуємо впровадити систему замовлення результатів. Вона буде полягати в наступному, коли користувач віддає дані застосунку, він віддає у відповідь користувачу унікальний код операції і приблизний час її завершення, водночас на стороні бекенду буде запущено обчислення. Коли обчислення завершиться, користувач може в спеціальному вікні / полі вводу ввести відповідний код і побачити результати. Результати будуть зберігатися протягом кількох днів, якщо користувач не зазначив свого бажання зберігати їх довше.

Попри на те, що подібний функціонал буде вимагати зберігання даних, ми все одно не будемо потребувати програмного сервера, оскільки дані можуть зберігатися в базі даних, що є окремим AWS сервісом, а з базою можуть працювати і AWS Lambda. Хоча впровадження бази даних і збільшить кошт утримання системи.

## 6.3. ПОКРАЩЕННЯ АНАЛІЗУ ПОГОДНИХ ДАНИХ

В майбутньому плановано розпочати співпрацю з дослідниками у галузі дослідження клімату, щоб краще дізнатися їх методи аналізу погодних даних, які процеси можна автоматизувати, які дані і в якій формі вони хотіли би бачити. Також треба продовжити вивчення алгоритмів машинного навчання, таких як t-SNE, з метою їх більш ефективного використання, а також кращого аналізу зображень, отриманих за допомогою їх.

## 6.4. ПОЧАТИ ПРЯМУ СПІВПРАЦЮ З КОМАНДОЮ WORLD WEATHER ONLINE

На жаль, безкоштовний режим користування World Weather Online API обмежений у часі (60 днів з дня початку дії пробної ліцензії). Тому, якщо проект надалі буде розвиватися, то співпраці з цією командою на ліцензійній основі не можна уникнути. В додаток до цього, під час такої співпраці завжди можна висловити свої побажання стосовно самого API на правах клієнта.

## 6.5. ПОКРАЩЕННЯ ФРОНТЕНДУ

Частино, що найдинамічніше з усіх розвивається, є фронтенд. В ньому будуть великі зміни, оскільки ми плануємо перехід з React.js/Redux до Angular 5, що дозволить будувати більш складний інтерфейс користувача.

Також, було би добре переробити компонент, що відповідає за введення періодів з двох окремих, погано пов’язаних полів в один компонент, що має вигляд єдиного компонента, поля вводу, тощо. До мапи можна додати елемент пошуку локації за її назвою.

# ВИСНОВКИ

Метою дипломної роботи було створення веб-застосунку, що дозволить користувачу візуалізувати введені ним багатовимірні дані. Щоб досягти цю мету, ми визначили наступні завдання дипломної роботи:

1. Ознайомитись з методиками візуалізації багатовимірних даних, зокрема алгоритмом T-SNE.
2. Визначити досліджувану множину даних.
3. Розробити архітектуру застосунку.
4. Обрати технології, за допомогою яких буде реалізовано застосунок.
5. Реалізувати застосунок.

Ми ознайомилися з алгоритмами та методами візуалізації багатовимірних даних, їх перевагами, недоліками та особливостями. Оскільки найбільшу зацікавленість в нас викликав алгоритм t-SNE (t-distributed stochastic neighbor embedding), тому було вирішено використовувати його за основу нашого застосунку. Попри його математичну складність, як самого алгоритму, так і його імплементації, нам вдалося не тільки працювати з ним, використовуючи готовий node.js модуль в режимі «чорної скриньки», але й ознайомитись з ним і максимально спрощено, але в той самий час точно, описати принцип дії алгоритму t-SNE.

Також, ми обрали конкретну предметну галузь, до якої буде застосований наш програмний продукт, а саме галузь спостереження за змінами клімату. Ця галузь в наш час є актуальною як ніколи, бо вже багато років науковці фіксують не тільки зміну загального клімату на планеті, а й прискорення цих деструктивних процесів. Саме тому, завдяки нашій програмі, ми захотіли зробити свій вклад в дослідження змін клімату, а також розвитку інструментів для подібних досліджень. Були визначені характеристики цих даних на кожному етапі обробки, від їх вводу користувачем, отримання даних з World Weather Online API до зниження їх розмірності і зображення даних на графіку.

Було вирішено спроектувати програмний продукт у формі веб-застосунку. Такий підхід надає доступ до застосунку всім користувачам мережі інтернет, попри операційну систему та характеристики їх обчислювального пристрою. Що безперечно забезпечує кросплатформеність застосунку і потенційно збільшує його аудиторію, в порівнянні з застосунками, виконаними з використанням підходу програмного забезпечення, що потребує встановлення на комп’ютері чи мобільному пристрої.

Як і кожен веб-застосунок, наш програмний продукт має в своєму складі клієнтську програму, що виконується на стороні браузера. Вона відповідає за взаємодію з користувачем, здійснення запитів до REST API ресурсів, збір даних від користувача, а також виведення результатів, що прийшли як відповіді з серверної програми. Також він має серверну програму, що виконує всі необхідні обчислення над даними і їх перетворення, і запити з метою отримання додаткових даних для формування коректної відповіді, що можна буде відправити до фронтенду. Хочеться зазначити особливий успіх нашої архітектури, яка не потребує для свого функціювання програмного сервера, а отже і віртуальної машини AWS EC2 на сторону серверу. Фронтенду використовує можливості AWS S3 Bucket з хостінгу та сервування своїх статичних сторінок, а бекенд використовує сервіс AWS Lambda, що дозволяє співвідносити запити до певної адреси з функцією, що буде виконуватись при кожному запитів і буде його оброблювати. Через те, що на відміну від AWS EC2, AWS Lambda виконується не весь час, а лише в відповідь на запити, ціна утримання такого програмного коду надзвичайно низька, а в нашому конкретному випадку, утримання цього коду безкоштовне, оскільки поки що, ми не перейшли ліміт 1 мільйон запитів на місяць.

Під час розробки веб-застосунку було використано велику кількість технологій, в тому числі, але не обмежуючись:

* AWS Lambda – подійно-орієнтована, безсерверна обчислювальна платформа, що була надана компанією Amazon як частина Amazon Web Services.
* AWS S3 Bucket – сервіс-сховище даних, один з Amazon Web Services.
* React.js – відкрита JavaScript бібліотека для створення інтерфейсів користувача, яка покликана вирішувати проблеми часткового оновлення вмісту веб-сторінки, з якими стикаються в розробці односторінкових застосунків. Розробляється Facebook, Instagram і спільнотою індивідуальних розробників.
* Redux - відкрита JavaScript бібліотека призначена для управління станом програми. Найчастіше використовується разом з React або Angular для побудови користувацьких інтерфейсів.
* Webpack – інструмент збірки фронтенд проекту. Допомагає зібрати React.js/Redux проект у працездатний веб-застосунок, який може бути виконаний браузером.

Загалом, розглянувши наш застосунок, без зайвої скромності, ми можемо стверджувати, що вдався дуже добрий програмний продукт, що виконує поставлені перед ним завдання, є приємний з естетичної точки зору, а також має інтуїтивно зрозумілий інтерфейс користувача, через що він не потребує додаткового інструктажу, щоб застосовувати програму у своїй дослідницькій діяльності. Не будемо приховувати, що наш програмний продукт не ідеальний, і до списку його недоліків можна віднести:

* обмеження по довжині окремого періоду, за який можна взяти погодні дані, через велику складність обчислень і технічних обмежень, які на нас накладають сервіси AWS Lambda і AWS Gateway API.
* відсутність системи автоматичного тестування і системи continuous integration, що в майбутньому ускладнить розробку, якщо обсяг проекту збільшиться
* покращення механізму аналізу даних і нагальна проблема встановлення контакту зі спеціалістами в галузі аналізу змін клімату, екологами є життєво необхідним для подальшого розвитку проекту
* зміна дизайну інтерфейсу користувача з естетичних мотивів (особиста думка автора дипломної роботи) .

Однак, слід пам’ятати про те, що всі ці пункти можна підписати не тільки «недоліки проекту», а ще й «напрямки розвитку проекту». Саме тому, ми вважаємо, що мета проекту була досягнута. Ми отримали корисний програмний продукт, що є динамічним і може бути далі покращеним, і може привести до продуктивної співпраці між багатьма колективами розробників з різними ідеями та корпоративними культурами, що в своїй сумі дасть унікальний результат.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. L.J.P. van der Maaten. Accelerating t-SNE using Tree-Based Algorithms. // Journal of Machine Learning Research. – 2014. – №15. – С. 3221–3245.
2. L.J.P. van der Maaten and G.E. Hinton. Visualizing Non-Metric Similarities in Multiple Maps. // Machine Learning. – 2012. – №87. – С. 33 – 55.
3. Learning a Parametric Embedding by Preserving Local Structure: матеріали конференції the Twelfth International Conference on Artificial Intelligence & Statistics (AI-STATS) / L.J.P. van der Maaten, квітень 16 – 18, 2009, Hilton Clearwater Beach Resort, Clearwater Beach, Florida USA.
4. L.J.P. van der Maaten and G.E. Hinton. Visualizing High-Dimensional Data Using t-SNE. // Journal of Machine Learning Research. – 2008. – №9 – С. 2579 – 2605.
5. Andreas Müller-Cyran. Introduction to Machine Learning with Python: A Guide for Data Scientists. / Andreas Müller-Cyran, Sarah Guido. – К.: O’Reilly, 2016 – 378 c.
6. Nikhil Buduma. Fundamentals of Deep Learning. / Nikhil Buduma. – K.: O’Reilly, 2017 – 283 c.
7. Josh Patterson. Deep Learning: A Practitioner's Approach / Josh Patterson. – K.: O’Reilly, 2017, – 507 c.
8. Peter Harrington. Machine Learning in Action. / Peter Harrington. – K.: Manning, 2012, – 354 c.
9. Drew Conway. Machine Learning for Hackers. / Drew Conway, John Myles White. – K.: O’Reilly, 2012 – 324 c.
10. Інформація про алгоритм t-SNE: [Електронний ресурс] // Офіційний сайт одного з авторів алгоритму L.J.P. van der Maaten, М., 2018, URL: <https://lvdmaaten.github.io/tsne/>. (Дата звернення 13.02.18)
11. Інструкція з використання імплементації алгоритму t-SNE для Node.js: [Електронний ресурс] // офіційний сайт менеджера пакетів до Node.js npm, М., 2018, URL: <https://www.npmjs.com/package/tsne-js>. (Дата звернення: 15.02.18)
12. Joshua Starmer. StatQuest: t-SNE, Clearly Explained : [Електронний ресурс] // Youtube, М., 2018, URL: <https://www.youtube.com/watch?v=NEaUSP4YerM>. (Дата звернення: 16.01.18)

# ДОДАТОК А.

Початковий код веб-застосунку «Кліматичний аналізатор».

1. Чо́рний я́щик (чорна скринька) — це термін, який використовується у техніці й кібернетиці для позначення об'єкта чи системи, про принципи дії яких нічого невідомо, крім того, що певному вхідному сигналу відповідає певний вихідний сигнал. [↑](#footnote-ref-1)
2. In science, computing, and engineering, a black box is a device, system or object which can be viewed in terms of its inputs and outputs (or transfer characteristics), without any knowledge of its internal workings. Its implementation is "opaque" (black). Almost anything might be referred to as a black box: a transistor, an algorithm, or the human brain. [↑](#footnote-ref-2)
3. Складне математичне поняття, що для наших цілей позначає різницю між двома ймовірнісними розподілами [↑](#footnote-ref-3)
4. редюсер – функція, що відповідає за зміну певної порції стану [↑](#footnote-ref-4)
5. дія – поняття в Redux, що позначає об’єкт, який оброблює редюсер [↑](#footnote-ref-5)
6. Promise – один з методів асинхронного програмування в JavaScript, що полягає в створенні об’єкту відповідного типу, і в залежності від його стану, виконання функцій, що були йому передані. [↑](#footnote-ref-6)